AIP-5SSB0

Adaptive Information Processing

Lecture Notes

Bert de Vries
Tjalling Tjalkens (Flux 7.101)
Marco Cox (Flux 7.060)

- Course Outline and Administrative Issues 6
  - Adaptive Information Processing (5SSB0) 6
  - AIP Logistic Issues 6
  - Course comes in Two Parts 6
  - Why Take This Class? 6
  - Materials 7
  - Exam Guide 7

- Machine Learning Overview 8
  - Preliminaries 8
  - What is Machine Learning? 8
  - Machine learning and the scientific inquiry loop. 8
  - Machine Learning is Difficult 9
  - A Machine Learning Taxonomy 9
  - Supervised Learning 10
  - Unsupervised Learning 10

- Probability Theory Review 12
  - Preliminaries 12
  - Example Problem: Disease Diagnosis 12
  - Why Probability Theory? 12
  - Probability Theory Notation 12
  - Probability Theory Calculus 13
  - Frequentist vs. Bayesian Interpretation of Probabilities 14
  - The Sum Rule and Marginalization 14
  - The Product Rule and Bayes Rule 15
  - Bayes Rule Nomenclature 15
  - The Likelihood Function vs the Sampling Distribution 15
  - Probabilistic Inference 16
  - Working out the example problem: Disease Diagnosis 17
  - Inference Exercise: Bag Counter 17
  - Inference Exercise: Causality? 17
  - PDF for the Sum of Two Variables 17
  - Expectation and Variance 18
  - Example: Mean and Variance for the Sum of Two Variables 19
  - Linear Transformations 19
  - Review Probability Theory 19

- Bayesian Machine Learning 20
  - Preliminaries 20
  - Example Problem: Predicting a Coin Toss 20
  - The Bayesian Machine Learning Framework 20
  - (1) Model specification 20
  - (2) Parameter estimation 21
  - (3) Prediction 21
  - Bayesian Model Comparison 21
  - Bayesian Model Comparison (continued) 22
  - Machine Learning and the Scientific Method Revisited 22
  - Now Solve the Example Problem: Predicting a Coin Toss 23
  - Coin toss example (1): Model Specification 23
  - Coin toss example (2): Parameter estimation 23
  - Coin Toss Example (3): Prediction 23
  - Coin Toss Example: What did we learn? 24
  - From Posterior to Point-Estimate 25
  - Some Well-known Point-Estimates 25
  - Bayesian vs Maximum Likelihood Learning 26
  - Report Card on Maximum Likelihood Estimation 26

- Working with Gaussians 27
    - Preliminaries 27
    - Sums and Transformations of Gaussian Variables 27
    - Example: Gaussian Signals in a Linear System 27
    - Example: Bayesian Estimation of a Constant 28
    - CODE EXAMPLE 29
    - Multivariate Gaussian Multiplication 29
    - Conditioning and Marginalization of a Gaussian 30
    - Example: Conditioning of Gaussian 31
    - Application: Recursive Bayesian Estimation 32
    - Product of Normally Distributed Variables 34
    - Review Gaussians 35
    - Some Useful Matrix Calculus 35
    - What's Next? 36
- Density Estimation 37
    - Preliminaries 37
    - Why Density Estimation? 37
    - Example Problem 37
    - Log-Likelihood for a Multivariate Gaussian (MVG) 38
    - Maximum Likelihood estimation of mean of MVG 39
    - Maximum Likelihood estimation of variance of MVG 39
    - Sufficient Statistics 39
    - Solution to Example Problem 40
    - Discrete Data: the 1-of-K Coding Scheme 40
    - Categorical vs. Multinomial Distribution 41
    - Maximum Likelihood Estimation for the Multinomial 41
    - Recap ML for Density Estimation 41
- Linear Regression 43
    - Preliminaries 43
    - Regression - Illustration 43
    - Regression vs Density Estimation 43
    - 1. Model Specification for Linear Regression 44
    - 2. ML Estimation for Linear Regression Model 44
    - 3. Prediction of New Data Points 44
    - Deterministic Least-Squares Regression 45
    - Probabilistic vs. Deterministic Approach 45
    - Not Identically Distributed Data 45
    - Too Few Training Samples 47
    - Too Few Training Samples, cont'd: the MAP estimate 47
    - Adaptive Linear Regression 47
- Generative Classification 49
    - Preliminaries 49
    - Example Problem: an apple or a peach? 49
    - Generative Classification Problem Statement 50
    - 1 - Model specification 51
    - 2 - Parameter Inference for Classification 51
    - 3 - Application: Class prediction for new Data 52
    - Discrimination Boundaries 52
    - Recap Generative Classification 54
- Discriminative Classification 55
    - Preliminaries 55
    - Problem: difficult class-conditional data distribitions 55
    - Main Idea of Discriminative Classification 56
    - 1. Model Specification 57
    - 2. ML Estimation for Discriminative Classification 57
    - 2. (OPTIONAL) Proof of Derivative of Log-likelihood for Discriminative Classification 57
    - 3. Application - Classify a new input 58
    - Recap Classification 59

- Clustering with Gaussian Mixture Models 60
  - Preliminaries 60
  - Limitations of Simple IID Gaussian Models 60
  - Towards More Flexible Models 60
  - Illustrative Example 61
  - Unobserved Classes 61
  - The Gaussian Mixture Model (GMM) 61
  - Gaussian Mixture Models 62
  - Inference: Log-Likelihood for GMM 62
  - Introducing a Soft Class Indicator 62
  - ML estimation for Clustering: The Expectation-Maximization (EM) Algorithm Idea 63
  - Clustering vs. (Generative) Classification 65

- The General EM Algorithm 66
  - Preliminaries 66
  - The Kullback-Leibler Divergence 66
  - EM by maximizing a Lower Bound on the Log-Likelihood 67
  - EM Details 68
  - Hoemwork exercise: EM for GMM Revisited 68
  - Homework Exercise: EM for Three Coins problem 68
  - Report Card on EM 68
  - (OPTIONAL SLIDE) The Free-Energy Principle 69
  - (OPTIONAL SLIDE) The Free-Energy Principle, cont'd 69
  - (OPTIONAL SLIDE) The Free-Energy Principle, cont'd 70

- Continuous Latent Variable Models - PCA and FA 71
  - Preliminaries 71
  - Continuous Latent Variable Models 71
  - Dimensionality Reduction 71
  - Example Problem: Visualization with missing data 72
  - Model Specification for LC-LVM 72
  - LC-LVM Analysis (1): The marginal distribution $p({x})#1 73
  - The Covariance Matrix of $p(x)$ is of Intermediate Complexity 73
  - LC-LVM Analysis (2): The Factor Loading Matrix W is Not Unique 73
  - LC-LVM analysis (3): Constraints on the Noise Variance $\Psi#1 74
  - LC-LVM analysis (3): Constraints on the Noise Variance Ψ, cont'd 74
  - Typical Applications 74
  - ML estimation for pPCA Model 75
  - Solution method 1: Gradient-ascent on the log-likelihood 75
  - Solution method 2: Use EM 75
  - Solution method 2: Use EM, cont'd 76
  - Example Problem Revisited 76

- Factor Graphs 79
  - Preliminaries 79
  - Why Factor Graphs? 79
  - Factor Graph Construction Rules 79
  - Some FFG Terminology 80
  - Equality Nodes for Branching Points 80
  - Equality Nodes for Branching Points, cont'd 80
  - Probabilistic Models as Factor Graphs 81
  - Inference by Closing Boxes 81
  - Evaluating the closing-the-box rule for individual nodes 83
  - Sum-Product Algorithm 83
  - Processing Observations in a Factor Graph 84
  - Example: SP Messages for the Equality Node 85
  - (OPTIONAL SLIDE) Example: SP Messages for the Addition Nodes 86
  - (OPTIONAL SLIDE) Example: SP Messages for Multiplication Nodes 86
  - Example: Bayesian Linear Regression 87
  - Homework Exercises 90
  - Inference in Linear Gaussian Models by Sum-Product Message Passing 91

- Dynamic Latent Variable Models 92
  - Preliminaries 92
  - Example Problem 92
  - Dynamical Models 92
  - State-space Models 93
  - Hidden Markov Models and Linear Dynamical Systems 94
  - Kalman Filtering 94
  - Message Passing in State-space Models 95
  - Example Problem Revisited 96
  - Extensions 97

- EM as a Message Passing Algorithm 98
    - Preliminaries 98
    - A Problem for the Multiplier Node 98
    - Limitations of Sum-Product Messages 99
    - EM as Message Passing 99
    - EM vs SP and MP Message Passing 101
    - A Snag for EM Message Passing on Deterministic Nodes 101
    - A Solution for the Multiplier Node with Unknown Coefficient 102
    - Automating Inference 103
    - Example: Linear Dynamical Systems 103

Course Outline and Administrative Issues

Adaptive Information Processing (5SSB0)

AIP Logistic Issues

- **When**: 3rd quartile, at 8 weeks of 4 hours per week.
- **Load**: Total workload is 5 ECTS $\Rightarrow 5 \times 28[\mathrm{hrs/ECTS}] = 140$ hours or $140/32 \approx 4.4$ study hours per lecture.
- **Web**: http://bertdv.github.io/teaching/AIP-5SSB0/ (or goto teaching tab at http://bertdv.nl)
- **Feedback**: The source materials for these lecture notes are at https://github.com/bertdv/AIP-5SSB0 . Please file a github issue if something is wrong or just unclear in these notes.

Course comes in Two Parts

Part I: Linear Gaussian Models and the EM Algorithm

- $4 \times 4 = 16$ lecture hours, mostly in February
- Instructor: Bert de Vries, rm. FLUX-7.060 (on Wednesdays)
- email bdevries@gnresound.com
- Code examples in Julia
  - Consult the **teaching assistant** Marco Cox (FLUX-7.060; email mail@marcocox.com) for help on Julia.

Part II: Model Complexity Control and the MDL Principle

- $4 \times 4 = 16$ lecture hours, mostly in March
- Instructor: Tjalling Tjalkens, rm. FLUX-7.101
- email t.j.tjalkens@tue.nl

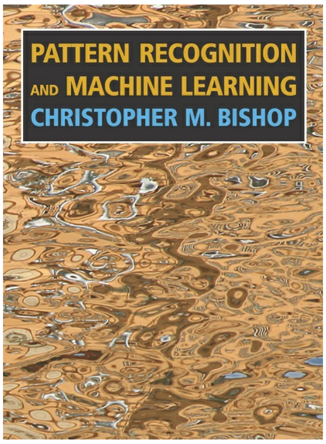
Why Take This Class?

- Suppose you need to develop an algorithm for a complex DSP task. This is what you'll do:
  1. Choose a set of candidate algorithms $y = H_k(x;\theta)$ where $k \in \{1,2,\dots,K\}$ and $\theta \in \Theta_k$; (you think that) there's an algorithm $H_{k^*}(x;\theta^*)$ that performs according to your liking.
  2. You collect a set of examples $D = \{(x_1,y_1),(x_2,y_2),\dots,(x_N,y_N)\}$ that are consistent with the correct algorithm behavior.

- Using the methods from this class, you will be able to design a suitable algorithm through:
  1. **model selection**, i.e., find k^* (mostly discussed in part 2 of this class)
  2. **parameter estimation**, i.e., find θ^* (mostly in part 1 of this class)

Materials

- Book (bol.com link):



- Background reading; covers about the same stuff as (mandatory) slides.
- Contains much more material; great for future study and reference.

Exam Guide

- Tested material consists of these lecture notes, reading assignments (as assigned in the first cell/slide of each lecture notebook) and exercises (see class website).
- Slides that are not required for the exam are indicated by the word (OPTIONAL) in the header.
- Advice: download (and make free use of) Sam Roweis' cheat sheets for Matrix identities and Gaussian formulas.
- **Very strong advice**: Make old exams!
- You are not allowed to use books nor bring printed or handwritten formula sheets to the exam. Difficult-to-remember formulas are supplied at the exam sheet (see old exams).
- You may use a simple pocket calculator, but no smartphones (only arithmetic assistance is allowed.)

Machine Learning Overview

Preliminaries

- Goal
  - Top-level overview of machine learning
- Materials
  - Study Bishop pp. 1-4
  - Study this notebook

What is Machine Learning?

- Machine Learning relates to **building models from data**.
  - Suppose we want to make a model for a complex process about which we have little knowledge (so hand-programming is not possible).

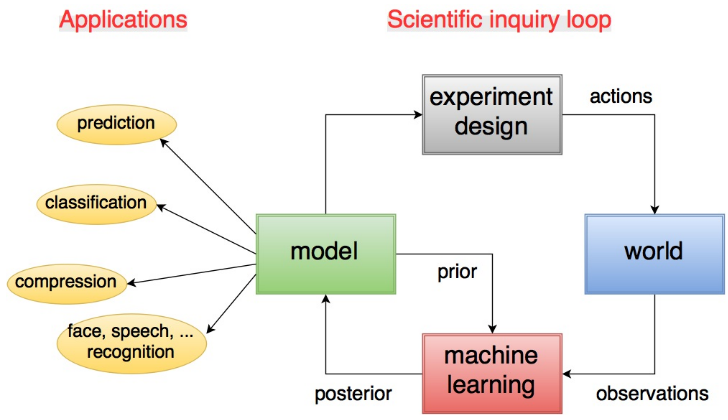
- **Solution**: Get the computer to program itself by showing it examples of the behavior that we want.

- Practically, we choose a library of models, and write a program that picks a model and tunes it to fit the data.

- **Criterion**: a *good* model generalizes well to unseen data from the same process.

- This method is known in various scientific communities under different names such as machine learning, statistical inference, system identification, data mining, source coding, data compression, etc.

Machine learning and the scientific inquiry loop.



Machine Learning is Difficult

- Modeling (Learning) Problems
  - Is there any regularity in the data anyway?
  - What is our prior knowledge and how to express it mathematically?
  - How to pick the model library?
  - How to tune the models to the data?
  - How to measure the generalization performance?

- Quality of Observed Data
  - Not enough data
  - Too much data?
  - Available data may be messy (measurement noise, missing data points, outliers)

A Machine Learning Taxonomy

- **Supervised Learning**: Given examples of inputs and corresponding desired outputs, predict outputs on future inputs.
  - Examples: classification, regression, time series prediction

- **Unsupervised Learning**: (a.k.a. **density estimation**). Given only inputs, automatically discover representations, features, structure, etc.
  - Examples: clustering, outlier detection, compression

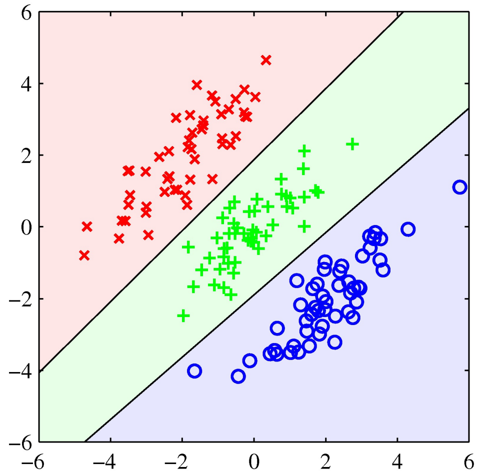
- **Reinforcement Learning**: Given sequences of inputs, actions from a fixed set, and scalar rewards/punishments, *learn* to select action sequences in a way that maximizes expected reward, e.g. chess and robotics. (This is more akin to learning how to design good experiments and is not covered in this course.)

- Other stuff, like **Preference Learning**, **learning to rank**, etc. (also not covered in this course). Note that many machine learning problems can be (re-)formulated as special cases of either a supervised or unsupervised problem, which are both covered in this class.

Supervised Learning

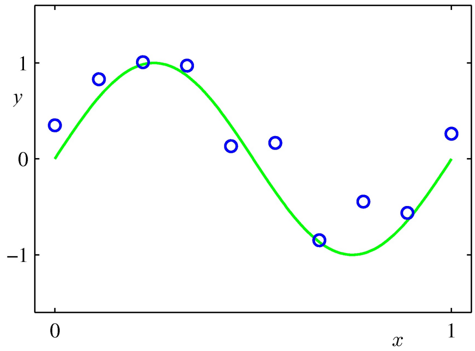
- Given observations $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, the goal is to estimate the conditional distribution $p(y|x)$.

Classification



- The target variable y is a *discrete-valued* vector representing class labels

Regression

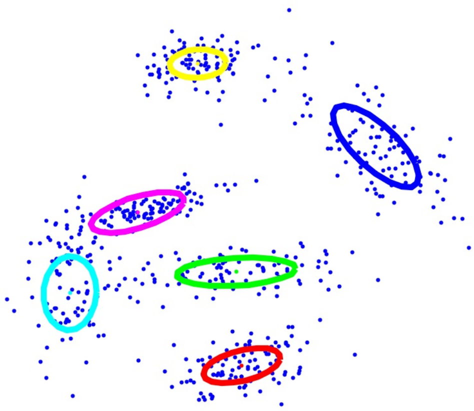


- Same problem statement as classification but now the target variable is a *real-valued* vector.

Unsupervised Learning

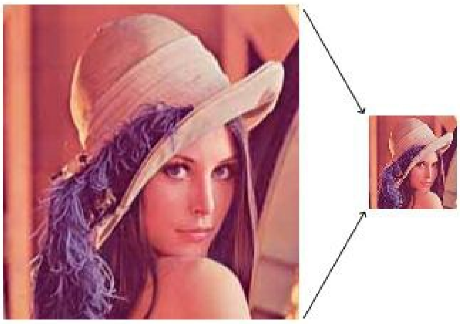
Given data $D = \{x_1, \ldots, x_N\}$, model the (unconditional) probability distribution $p(x)$ (a.k.a. **density estimation**).

Clustering



- Group data into clusters such that all data points in a cluster have similar properties.

Compression / dimensionality reduction



- Output from coder is much smaller in size than original, but if coded signal if further processed by a decoder, then the result is very close (or exactly equal) to the original.

Probability Theory Review

Preliminaries

- Goal
  - Review of probability theory from a logical reasoning viewpoint (i.e., a Bayesian interpretation)
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 12-20
    - Bruininkx - 2002 - Bayesian Probability

Example Problem: Disease Diagnosis

- **[Question]** Given a disease with prevalence of 1% and a test procedure with sensitivity ('true positive' rate) of 95% and specificity ('true negative' rate) of 85% , what is the chance that somebody who tests positive actually has the disease?

- **[Solution]** Use probabilistic inference, to be discussed in this lecture.

Why Probability Theory?

- Probability theory (PT) is the **theory of optimal processing of incomplete information** (see Cox theorem), and as such provides a quantitative framework for drawing conclusions from a finite (read: incomplete) data set.

- Machine learning concerns drawing conclusions from (a finite set of) data and therefore PT is the *optimal calculus for machine learning*.

- In general, nearly all interesting questions in machine learning can be stated in the following form (a conditional probability):
$$p(\text{whatever-we-want-to-know} \mid \text{whatever-we-do-know})$$
  - For example:
    - Predictions
$$p(\,\text{future-observations} \mid \text{past-observations}\,)$$
    - Classify a received data point
$$p(\,x\text{-belongs-to-class-}k \mid x\,)$$

- **Information theory** ("theory of log-probability") provides a source coding view on machine learning that is consistent with probability theory (more in part-2).

Probability Theory Notation

- Define an **event** A as a statement, whose truth is contemplated by a person, e.g.,

$$A = \text{'it will rain tomorrow'}$$

- We write the denial of A, i.e. the event **not**-A, as $\bar{A}$.

probabilities

- For any event A, with background knowledge I, the **conditional probability of A given I**, is written as
$$p(A|I)$$
- The value of a probability is limited to $0 \leq p(A|I) \leq 1$.
- All probabilities are in principle conditional probabilities of the type $p(A|I)$, since there is always some background knowledge.
- Still, we often write $p(A)$ rather than $p(A|I)$ if the background knowledge I is assumed to be obviously present. E.g., $p(A)$ rather than $p(\,A \,|\, \text{the-sun-comes-up-tomorrow}\,)$.

probabilities for random variable assignments

- Note that, if X is a random variable, then the assignment $X = x$ (with x a value) can be interpreted as an event.
- We often write $p(x)$ rather than $p(X = x)$ (hoping that the reader understands the context ;-)
- In an apparent effort to further abuse notational conventions, $p(X)$ often denotes the full distribution over random variable X, i.e., the distribution for all assignments for X.

compound events

- The **joint** probability that both A and B are true, given I (a.k.a. **conjunction**) is written as
$$p(A, B|I)$$
- A and B are said to be **independent**, given I, if (and only if)
$$p(A, B|I) = p(A|I)\, p(B|I)$$
- The probability that either A or B, or both A and B, are true, given I (a.k.a. **disjunction**) is written as
$$p(A + B|I)$$

Probability Theory Calculus

- **Normalization**. If you know that event A given I is true, then $p(A|I) = 1$.
- **Product rule**. The conjuction of two events A and B with given background I is given by
$$p(A, B|I) = p(A|B, I)\, p(B|I)\,.$$
  - If A and B are independent given I, then $p(A|B, I) = p(A|I)$.
- **Sum rule**. The disjunction for two events A and B given background I is given by
$$p(A + B|I) = p(A|I) + p(B|I) - p(A, B|I)\,.$$
  - As a special case, it follows from the sum rule that $p(A|I) + p(\bar{A}|I) = 1$
  - Note that the background information may not change, e.g., if $I' \neq I$, then
$$p(A + B|I') \neq p(A|I) + p(B|I) - p(A, B|I)\,.$$

- **All legitimate probabilistic relations can be derived from the sum and product rules!**

- The product and sum rules are also known as the **axioms of probability theory**, but in fact, under some mild conditions, they can be derived as the unique rules for rational reasoning under uncertainty (Cox theorem, 1946).

Frequentist vs. Bayesian Interpretation of Probabilities

- In the **frequentist** interpretation, $p(A)$ relates to the relative frequency that A would occur under repeated execution of an experiment.

- For instance, if the experiment is tossing a coin, then $p(\texttt{tail}) = 0.4$ means that in the limit of a large number of coin tosses, 40% of outcomes turn up as `tail`.

- In the **Bayesian** interpretation, $p(A)$ reflects the **degree of belief** that event A is true. I.o.w., the probability is associated with a **state-of-knowledge** (usually held by a person).
  - For instance, for the coin tossing experiment, $p(\texttt{tail}) = 0.4$ should be interpreted as the belief that there is a 40% chance that `tail` comes up if the coin were tossed.
  - Under the Bayesian interpretation, PT calculus (sum and product rules) **extends boolean logic to rational reasoning with uncertainty**.

- The Bayesian viewpoint is more generally applicable than the frequentist viewpoint, e.g., it is hard to apply the frequentist viewpoint to the event '`it will rain tomorrow`'.

- The Bayesian viewpoint is clearly favored in the machine learning community. (In this class, we also strongly favor the Bayesian interpretation).

The Sum Rule and Marginalization

- We discussed that every inference problem in PT can be evaluated through the sum and product rules. Next, we present two useful corollaries: (1) Marginalization and (2) Bayes rule

- If X and Y are random variables over finite domains, than it follows from the sum rule that
$$p(X) = \sum_Y p(X,Y) = \sum_Y p(X|Y)p(Y) \,.$$

- Note that this is just a **generalized sum rule**. In fact, Bishop (p.14) (and some other authors as well) calls this the sum rule.
  - EXERCISE: Proof the generalized sum rule.

- Of course, in the continuous domain, the (generalized) sum rule becomes
$$p(X) = \int p(X,Y)\,\mathrm{d}Y$$

- Integrating Y out of a joint distribution is called **marginalization** and the result $p(X)$ is sometimes referred to as the **marginal probability**.

The Product Rule and Bayes Rule

- Consider 2 variables D and θ; it follows symmetry arguments that
$$p(D,\theta) = p(D|\theta)p(\theta) = p(\theta|D)p(D)$$
and hence that
$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}\,.$$

- This formula is called **Bayes rule** (or Bayes theorem). While Bayes rule is always true, a particularly useful application occurs when D refers to an observed data set and θ is set of model parameters that relates to the data. In that case,
  - the **prior** probability $p(\theta)$ represents our **state-of-knowledge** about proper values for θ, before seeing the data D.
  - the **posterior** probability $p(\theta|D)$ represents our state-of-knowledge about θ after we have seen the data.

$\Rightarrow$ Bayes rule tells us how to update our knowledge about model parameters when facing new data. Hence,

Bayes rule is the fundamental rule for machine learning!

Bayes Rule Nomenclature

- Some nomenclature associated with Bayes rule:
$$\underbrace{p(\theta|D)}_{\text{posterior}} = \frac{\overbrace{p(D|\theta)}^{\text{likelihood}} \times \overbrace{p(\theta)}^{\text{prior}}}{\underbrace{p(D)}_{\text{evidence}}}$$

- Note that the evidence (a.k.a. *marginal likelihood*) can be computed from the numerator through marginalization since
$$p(D) = \int p(D,\theta)\,\mathrm{d}\theta = \int p(D|\theta)\,p(\theta)\,\mathrm{d}\theta$$

- Hence, likelihood and prior is sufficient to compute both the evidence and the posterior. To emphasize that point, Bayes rule is sometimes written as
$$p(\theta|D)\,p(D) = p(D|\theta)\,p(\theta)$$

- For given D, the posterior probabilities of the parameters scale relatively against each other as
$$p(\theta|D) \propto p(D|\theta)p(\theta)$$

$\Longrightarrow$ All that we can learn from the observed data is contained in the likelihood function $p(D|\theta)$. This is called the **likelihood principle**.

The Likelihood Function vs the Sampling Distribution

- Consider a model $p(D|\theta)$, where D relates to a data set and θ are model parameters.

- In general, $p(D|\theta)$ is just a function of the two variables D and θ. We distinguish two interpretations of this function, depending on which variable is observed (or given by other means).

- The **sampling distribution** (a.k.a. the **data-generating** distribution)
$$p(D|\theta=\theta_0)$$
(a function of D only) describes the probability distribution for data D, assuming that it is generated by the given model with parameters fixed at $\theta = \theta_0$.

- In a machine learning context, often the data is observed, and θ is the free variable. For given observations $D = D_0$, the **likelihood function** (which is a function only of the model parameters θ) is defined as

$$\mathrm{L}(\theta) \triangleq p(D = D_0|\theta)$$

- Note that $\mathrm{L}(\theta)$ is not a probability distribution for θ since in general $\sum_\theta \mathrm{L}(\theta) \neq 1$.

- Technically, it is more correct to speak about the likelihood of a model (or model parameters) than about the likelihood of an observed data set. (Why?)

CODE EXAMPLE

Consider the following simple model for the outcome (head or tail) of a biased coin toss with parameter $\theta \in [0,1]$:

$$y \in \{0,1\}$$
$$p(y|\theta) \triangleq \theta^y (1-\theta)^{1-y}$$

We can plot both the sampling distribution (i.e. $p(y|\theta = 0.8)$) and the likelihood function (i.e. $L(\theta) = p(y = 0|\theta)$).

```
using Reactive, Interact, PyPlot
p(y,θ) = θ.^y .* (1-θ).^(1-y)
f = figure()
@manipulate for y=false, θ=0:0.1:1; withfig(f) do
        # Plot the sampling distribution
        subplot(221); stem([0,1], p([0,1],θ));
        title("Sampling distribution");
        xlim([-0.5,1.5]); ylim([0,1]); xlabel("y"); ylabel("p(y|θ=$(θ))");
        # Plot the likelihood function
        _θ = linspace(0.0, 1.0, 100)
        subplot(222); plot(_θ, p(convert(Float64,y), _θ));
        title("Likelihood function");
        xlabel("θ");
        ylabel("L(θ) = p(y=$(convert(Float64,y))|θ)");
    end
end
```

The (discrete) sampling distribution is a valid probability distribution. However, the likelihood function $L(\theta)$ clearly isn't, since $\int_0^1 L(\theta)\mathrm{d}\theta \neq 1$.

Probabilistic Inference

- **Probabilistic inference** refers to computing

$$p(\,\text{whatever-we-want-to-know} \,|\, \text{whatever-we-already-know}\,)$$

  - For example:

$$p(\,\text{Mr.S.-killed-Mrs.S.} \,|\, \text{he-has-her-blood-on-his-shirt}\,)$$
$$p(\,\text{transmitted-codeword} \,|\, \text{received-codeword}\,)$$

- This can be accomplished by repeated application of sum and product rules.

- For instance, consider a joint distribution $p(X,Y,Z)$. Assume we are interested in $p(X|Z)$:

$$p(X|Z) \stackrel{p}{=} \frac{p(X,Z)}{p(Z)} \stackrel{s}{=} \frac{\sum_Y p(X,Y,Z)}{\sum_{X,Y} p(X,Y,Z)} \,,$$

where the 's' and 'p' above the equality sign indicate whether the sum or product rule was used.

- In the rest of this course, we'll encounter many long probabilistic derivations. For each manipulation, you should be able to associate an 's' (for sum rule), a 'p' (for product or Bayes rule) or an 'a' (for a model assumption) above any equality sign. If you can't do that, file a github issue :)

Working out the example problem: Disease Diagnosis

- [Question] - Given a disease D with prevalence of 1% and a test procedure T with sensitivity ('true positive' rate) of 95% and specificity ('true negative' rate) of 85%, what is the chance that somebody who tests positive actually has the disease?

- [Answer] - The given data are $p(D=1)=0.01$, $p(T=1|D=1)=0.95$ and $p(T=0|D=0)=0.85$. Then according to Bayes rule,

$$\begin{aligned} p(D=1\,|T=1) &= \frac{p(T=1|D=1)p(D=1)}{p(T=1)} \\ &= \frac{p(T=1|D=1)p(D=1)}{p(T=1|D=1)p(D=1)+p(T=1|D=0)p(D=0)} \\ &= \frac{0.95\times 0.01}{0.95\times 0.01+0.15\times 0.99}=0.0601 \end{aligned}$$

Inference Exercise: Bag Counter

- [Question] - A bag contains one ball, known to be either white or black. A white ball is put in, the bag is shaken, and a ball is drawn out, which proves to be white. What is now the chance of drawing a white ball?

- [Answer] - Again, use Bayes and marginalization to arrive at $p(\text{white}|\text{data})=2/3$, see homework exercise
- $\Rightarrow$ Note that probabilities describe **a person's state of knowledge** rather than a 'property of nature'.

- [Excercise] - Is a speech signal a 'probabilistic' (random) or a deterministic signal?

Inference Exercise: Causality?

- [Question] - A dark bag contains five red balls and seven green ones. (a) What is the probability of drawing a red ball on the first draw? Balls are not returned to the bag after each draw. (b) If you know that on the second draw the ball was a green one, what is now the probability of drawing a red ball on the first draw?

- [Answer] - (a) $5/12$. (b) $5/11$, see homework.
- $\Rightarrow$ Again, we conclude that conditional probabilities reflect **implications for a state of knowledge** rather than temporal causality.

PDF for the Sum of Two Variables

- [Question] - Given two random **independent** variables X and Y, with PDF's $p_x(x)$ and $p_y(y)$. What is the PDF of $$Z=X+Y\,?$$

- [Answer] - Let $p_z(z)$ be the probability that Z has value z. This occurs if X has some value x and at the same time $Y=z-x$, with joint probability $p_x(x)p_y(z-x)$. Since x can be any value, we sum over all possible values for x to get
$$p_z(z)=\int_{-\infty}^{\infty}p_x(x)p_y(z-x)\,\mathrm{d}x$$
  - Iow, $p_z(z)$ is the **convolution** of p_x and p_y .

- Note that $p_z(z)\neq p_x(x)+p_y(y)$!!

- $\Rightarrow$ In linear stochastic systems theory, the Fourier Transform of a PDF (i.e., the characteristic function) plays an important computational role.

- This list shows how these convolutions work out for a few common probability distributions.

CODE EXAMPLE

- Consider the PDF of the sum of two independent Gaussians X and Y:

$$\begin{aligned} p_X(x) &= \mathcal{N}(\,x\,|\,\mu_X,\sigma_X^2\,) \\ p_Y(y) &= \mathcal{N}(\,y\,|\,\mu_Y,\sigma_Y^2\,) \\ Z &= X+Y \end{aligned}$$

- Performing the convolution (nice exercise) yields a Gaussian PDF for Z:

$$p_Z(z) = \mathcal{N}(\,z\,|\,\mu_X+\mu_Y,\sigma_X^2+\sigma_Y^2\,).$$

```
using Reactive, Interact, PyPlot, Distributions
f = figure()
@manipulate for μx=-2:0.1:2, σx=0.1:0.1:1.9,μy=0:0.1:4, σy=0.1:0.1:0.9; withfig(f) do
        μz = μx+μy; σz = sqrt(σx^2 + σy^2)
        x = Normal(μx, σx)
        y = Normal(μy, σy)
        z = Normal(μz, σz)
        range_min = minimum([μx-2*σx, μy-2*σy, μz-2*σz])
        range_max = maximum([μx+2*σx, μy+2*σy, μz+2*σz])
        range = linspace(range_min, range_max, 100)
        plot(range, pdf.(x,range), "k-")
        plot(range, pdf.(y,range), "b-")
        plot(range, pdf.(z,range), "r-")
        legend([L"p_X", L"p_Y", L"p_Z"])
        grid()
    end
end
```

Expectation and Variance

- The **expected value** or **mean** is defined as

$$\mathrm{E}[f] \triangleq \int f(x)\,p(x)\,\mathrm{d}x$$

- The **variance** is defined as

$$\mathrm{var}[f] \triangleq \mathrm{E}\left[(f(x)-\mathrm{E}[f(x)])^2\right]$$

- The **covariance** matrix between *vectors* x and y is defined as

$$\begin{aligned} \mathrm{cov}[x,y] &\triangleq \mathrm{E}\left[(x-\mathrm{E}[x])(y^T-\mathrm{E}[y^T])\right] \\ &= \mathrm{E}[xy^T]-\mathrm{E}[x]\mathrm{E}[y^T] \end{aligned}$$

    - Also useful as: $\mathrm{E}[xy^T] = \mathrm{cov}[x,y]+\mathrm{E}[x]\mathrm{E}[y^T]$

Example: Mean and Variance for the Sum of Two Variables

- For any distribution of x and y and $z = x + y$,

$$\begin{aligned}
\mathrm{E}[z] &= \int_z z \left[\int_x p_x(x) p_y(z-x) \,\mathrm{d}x \right] \,\mathrm{d}z \\
&= \int_x p_x(x) \left[\int_z z p_y(z-x) \,\mathrm{d}z \right] \,\mathrm{d}x \\
&= \int_x p_x(x) \left[\int_{y'} (y'+x) p_y(y') \,\mathrm{d}y' \right] \,\mathrm{d}x \\
&= \int_x p_x(x) \left(\mathrm{E}[y] + x \right) \,\mathrm{d}x \\
&= \mathrm{E}[x] + \mathrm{E}[y] \qquad \text{(always; follows from SRG-3a)}
\end{aligned}$$

- Derive as an exercise that

$$\begin{aligned}
\mathrm{var}[z] &= \mathrm{var}[x] + \mathrm{var}[y] + 2\mathrm{cov}[x,y] \qquad \text{(always, see SRG-3b)} \\
&= \mathrm{var}[x] + \mathrm{var}[y] \qquad \text{(if X and Y are independent)}
\end{aligned}$$

Linear Transformations

No matter how x is distributed, we can easily derive that **(do as exercise)**

$$\begin{aligned}
\mathrm{E}[Ax+b] &= A\mathrm{E}[x] + b \qquad &\text{(SRG-3a)} \\
\mathrm{cov}[Ax+b] &= A\,\mathrm{cov}[x]\,A^T \qquad &\text{(SRG-3b)}
\end{aligned}$$

- (The tag (SRG-3a) refers to the corresponding eqn number in Sam Roweis' Gaussian Identities notes.)

Review Probability Theory

- Interpretation as a degree of belief, i.e. a state-of-knowledge, not as a property of nature.

- We can do everything with only the **sum rule** and the **product rule**. In practice, **Bayes rule** and **marginalization** are often very useful for computing

$$p(\text{ what-we-want-to-know} \mid \text{what-we-already-know }) \,.$$

- Bayes rule

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}$$

  is the fundamental rule for learning!

- That's really about all you need to know about probability theory, but you need to *really* know it, so do the exercises.

Bayesian Machine Learning

Preliminaries

- Goals
  - Introduction to Bayesian (i.e., probabilistic) modeling
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 21-24

Example Problem: Predicting a Coin Toss

- **Question**. We observe a the following sequence of heads (h) and tails (t) when tossing the same coin repeatedly
$$D=\{hthhtth\}\,.$$
- What is the probability that heads (h) comes up next?

- **Answer** later in this lecture.

The Bayesian Machine Learning Framework

- Suppose that your task is to predict a 'new' datum x, based on N observations $D=\{x_1,\dots,x_N\}$.

- The Bayesian approach for this task involves three stages:
  1. Model specification
  2. parameter estimation (inference, learning)
  3. Prediction (apply the model)

- Next, we discuss these three stages in a bit more detail.

(1) Model specification

Your first task is to propose a model with tuning parameters θ for generating the data D.

- This involves specification of $p(D|\theta)$ and a prior for the parameters $p(\theta)$.

- *You* choose the distribution $p(D|\theta)$ based on your physical understanding of the data generating process.
  - Note that, for independent observations x_n ,
$$p(D|\theta)=\prod_{n=1}^{N}p(x_n|\theta)$$
so usually you select a model for generating one observation x_n and then use (in-)dependence assumptions to combine these models into a model for D.

- *You* choose the prior $p(\theta)$ to reflect what you know about the parameter values before you see the data D.

(2) Parameter estimation

- After model specification, you need to measure/collect a data set D. Then, use Bayes rule to find the posterior distribution for the parameters,

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} \propto p(D|\theta)p(\theta)$$

- Note that there's **no need for you to design a *smart* parameter estimation algorithm**. The only complexity lies in the computational issues.

- This "recipe" works only if the RHS factors can be evaluated; this is what machine learning is about
$\Rightarrow$ **Machine learning is easy, apart from computational details:)**

(3) Prediction

- Given the data D, our knowledge about the yet unobserved datum x is captured by

$$\begin{aligned} p(x|D) &= \int p(x,\theta|D) \,\mathrm{d}\theta \\ &= \int p(x|\theta,D)p(\theta|D) \,\mathrm{d}\theta \\ &= \int p(x|\theta)p(\theta|D) \,\mathrm{d}\theta \end{aligned}$$

- Again, **no need to invent a special prediction algorithm**. Probability theory takes care of all that. The complexity of prediction is just computational: how to carry out the marginalization over θ.

- In order to execute prediction, you need to have access to the factors $p(x|\theta)$ and $p(\theta|D)$. Where do these factors come from? Are they available?

- What did we learn from D? Without access to D, we would predict new observations through

$$p(x) = \int p(x,\theta) \,\mathrm{d}\theta = \int p(x|\theta)p(\theta) \,\mathrm{d}\theta$$

- NB The application of the learned posterior $p(\theta|D)$ not necessarily has to be prediction. We use it here as an example, but other applications are of course also possible.

Bayesian Model Comparison

- There appears to be a remaining problem: How good really were our model assumptions $p(x|\theta)$ and $p(\theta)$?

- Technically, this is a **model comparison** problem

- [**Q**.] What if I have more candidate models, say $\mathcal{M} = \{m_1,\ldots,m_K\}$ where each model relates to specific prior $p(\theta|m_k)$ and likelihood $p(D|\theta,m_k)$? Can we evaluate the relative performance of a model against another model from the set?

- [**A**.]: Start again with **model specification**. Specify a prior $p(m_k)$ for each of the models and then solve the desired inference problem:

$$
\begin{aligned}
p(m_k|D) &= \frac{p(D|m_k)p(m_k)}{p(D)} \\
&\propto p(m_k)\cdot p(D|m_k) \\
&= p(m_k)\cdot \int_\theta p(D,\theta|m_k)\,\mathrm{d}\theta \\
&= \underbrace{p(m_k)}_{\substack{\text{model}\\\text{prior}}}\cdot \int_\theta \underbrace{p(D|\theta,m_k)}_{\text{likelihood}}\,\underbrace{p(\theta|m_k)}_{\text{prior}}\,\mathrm{d}\theta
\end{aligned}
$$

Bayesian Model Comparison (continued)

- You, the engineer, have to choose the factors $p(D|\theta,m_k)$, $p(\theta|m_k)$ and $p(m_k)$. After that, for a given data set D, the model posterior $p(m_k|D)$ can be computed.

- If you need to work with one model,select the model with largest posterior $p(m_k|D)$

- Alternatively, if you don't want to choose a model, you can do prediction by **Bayesian model averaging** to utilize the predictive power from all models:

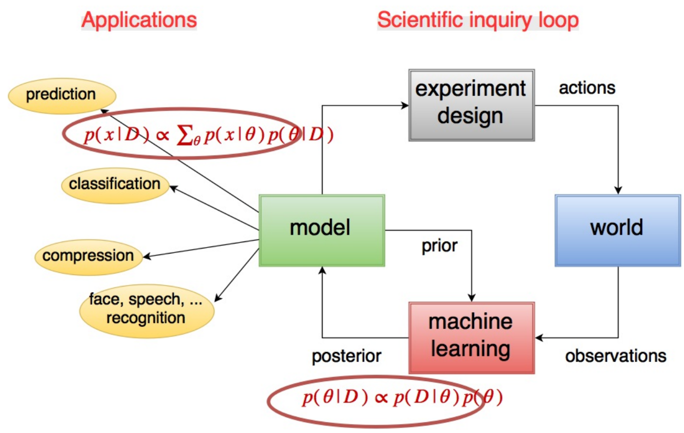
$$
\begin{aligned}
p(x|D) &= \sum_k \int p(x,\theta,m_k|D)\,\mathrm{d}\theta \\
&= \sum_k \underbrace{p(m_k|D)}_{\substack{\text{model}\\\text{posterior}}}\cdot \int \underbrace{p(\theta|D)}_{\substack{\text{parameter}\\\text{posterior}}}\,\underbrace{p(x|\theta,m_k)}_{\text{likelihood}}\mathrm{d}\theta
\end{aligned}
$$

- $\Rightarrow$ In a Bayesian framework, **model comparison** follows the same recipe as parameter estimation; it just works at one higher hierarchical level.

- More on this in part 2 (Tjalkens).

Machine Learning and the Scientific Method Revisited

- Bayesian probability theory provides a unified framework for information processing (and even the Scientific Method).



Now Solve the Example Problem: Predicting a Coin Toss

- We observe a the following sequence of heads (h) and tails (t) when tossing the same coin repeatedly

$$D=\{hthhtth\}\,.$$

- What is the probability that heads (h) comes up next? We solve this in the next slides ...

Coin toss example (1): Model Specification

We observe a sequence of N coin tosses $D=\{x_1,\ldots,x_N\}$ with n heads.

Likelihood

- Assume a Bernoulli distributed variable $p(x_k=h|\mu)=\mu$, which leads to a **binomial** distribution for the likelihood (assume n times heads were thrown):

$$p(D|\mu)=\prod_{k=1}^N p(x_k|\mu)=\mu^n(1-\mu)^{N-n}$$

Prior

- Assume the prior belief is governed by a **beta distribution**

$$p(\mu)=\mathcal{B}(\mu|\alpha,\beta)=\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\mu^{\alpha-1}(1-\mu)^{\beta-1}$$

- The Beta distribution is a **conjugate prior** for the Binomial distribution, which means that

$$\text{beta}\propto\text{binomial}\times\text{beta}$$

- α and β are called **hyperparameters**, since they parameterize the distribution for another parameter (μ). E.g., $\alpha=\beta=1$ (uniform).

Coin toss example (2): Parameter estimation

- Infer posterior PDF over μ through Bayes rule

$$\begin{aligned}p(\mu|D)&\propto p(D|\mu)\,p(\mu|\alpha,\beta)\\&=\mu^n(1-\mu)^{N-n}\times\mu^{\alpha-1}(1-\mu)^{\beta-1}\\&=\mu^{n+\alpha-1}(1-\mu)^{N-n+\beta-1}\end{aligned}$$

hence the posterior is also beta distributed as

$$p(\mu|D)=\mathcal{B}(\mu|\,n+\alpha,N-n+\beta)$$

- Essentially, **here ends the machine learning activity**

Coin Toss Example (3): Prediction

- Now, we want to **use** the trained model. Let's use it to predict future observations.

- Marginalize over the parameter posterior to get the predictive PDF for a new coin toss $x_\bullet$, given the data D,

$$\begin{aligned}
p(x_\bullet = h|D) &= \int_0^1 p(x_\bullet = h|\mu)\, p(\mu|D)\, \mathrm{d}\mu \\
&= \int_0^1 \mu \times \mathcal{B}(\mu|\, n+\alpha, N-n+\beta)\, \mathrm{d}\mu \\
&= \frac{n+\alpha}{N+\alpha+\beta} \qquad \text{(a.k.a. Laplace rule)}
\end{aligned}$$

- Finally, we're ready to solve our example problem: for $D = \{hthhtth\}$ and uniform prior ($\alpha = \beta = 1$), we get

$$p(x_\bullet = h|D) = \frac{n+1}{N+2} = \frac{4+1}{7+2} = \frac{5}{9}$$

Coin Toss Example: What did we learn?

- What did we learn from the data? Before seeing any data, we think that

$$p(x_\bullet = h) = p(x_\bullet = h|D)\big|_{n=N=0} = \frac{\alpha}{\alpha+\beta}\,.$$

- After the N coin tosses, we think that $p(x_\bullet = h|D) = \frac{n+\alpha}{N+\alpha+\beta}$.

- Note the following decomposition

$$\begin{aligned}
p(x_\bullet = h|\, D) &= \frac{n+\alpha}{N+\alpha+\beta} = \frac{n}{N+\alpha+\beta} + \frac{\alpha}{N+\alpha+\beta} \\
&= \frac{N}{N+\alpha+\beta}\cdot\frac{n}{N} + \frac{\alpha+\beta}{N+\alpha+\beta}\cdot\frac{\alpha}{\alpha+\beta} \\
&= \underbrace{\frac{\alpha}{\alpha+\beta}}_{prior} + \underbrace{\frac{N}{N+\alpha+\beta}}_{gain}\cdot\big(\underbrace{\frac{n}{N}}_{MLE} - \underbrace{\frac{\alpha}{\alpha+\beta}}_{prior}\big)
\end{aligned}$$

- Note that, since $0 \leq \text{gain} < 1$, the Bayesian estimate lies between prior and maximum likelihood estimate.

- For large N, the gain goes to 1 and $p(x_\bullet = h|D)$ goes to the maximum likelihood estimate (the relative frequency) n/N.

CODE EXAMPLE

Bayesian evolution of $p(\mu|D)$ for the coin toss

Let's see how $p(\mu|D)$ evolves as we increase the number of coin tosses N. We'll use two different priors to demonstrate the effect of the prior on the posterior (set $N = 0$ to inspect the prior).

```julia
using Reactive, Interact, PyPlot, Distributions
f = figure()
range = linspace(0,1,100)
μ = 0.4
samples = rand(192) .<= μ # Flip 192 coins
@manipulate for N=0:1:192; withfig(f) do
        n = sum(samples[1:N]) # Count number of heads in first N flips
        posterior1 = Beta(1+n, 1+(N-n))
        posterior2 = Beta(5+n, 5+(N-n))
        plot(range, pdf.(posterior1,range), "k-")
        plot(range, pdf.(posterior2,range), "k--")
        xlabel(L"\mu"); ylabel(L"p(\mu|\mathcal{D})"); grid()
        title(L"p(\mu|\mathcal{D})"*" for N=$(N), n=$(n) (real \$\\mu\$=$(μ))")
        legend(["Based on uniform prior "*L"B(1,1)","Based on prior "*L"B(5,5)"], loc=4)
    end
end
```

$\Rightarrow$ With more data, the relevance of the prior diminishes!

From Posterior to Point-Estimate

- Sometimes we want just one 'best' parameter (vector), rather than a posterior distribution over parameters. Why?
- Recall Bayesian prediction

$$p(x|D) = \int p(x|\theta)p(\theta|D)\,\mathrm{d}\theta$$

- If we approximate posterior $p(\theta|D)$ by a delta function for one 'best' value $\hat\theta$, then the predictive distribution collapses to

$$p(x|D) = \int p(x|\theta)\,\delta(\theta-\hat\theta)\,\mathrm{d}\theta = p(x|\hat\theta)$$

- This is the model $p(x|\theta)$ evaluated at $\theta=\hat\theta$.
- Note that $p(x|\hat\theta)$ is much easier to evaluate than the integral for full Bayesian prediction.

Some Well-known Point-Estimates

- **Bayes estimate**

$$\hat\theta_{bayes} = \int \theta\, p(\theta|D)\,\mathrm{d}\theta$$

- (homework). Proof that the Bayes estimate minimizes the expected mean-square error, i.e., proof that

$$\hat\theta_{bayes} = \arg\min_{\hat\theta} \int_\theta (\hat\theta-\theta)^2 p(\theta|D)\,\mathrm{d}\theta$$

- **Maximum A Posteriori** (MAP) estimate

$$\hat\theta_{\text{map}} = \arg\max_{\theta} p(\theta|D) = \arg\max_{\theta} p(D|\theta)\,p(\theta)$$

- **Maximum Likelihood** (ML) estimate

$$\hat\theta_{ml} = \arg\max_{\theta} p(D|\theta)$$

  - Note that Maximum Likelihood is MAP with uniform prior

Bayesian vs Maximum Likelihood Learning

Consider the task: predict a datum x from an observed data set D.

	Bayesian	**Maximum Likelihood**
1. **Model Specification**	Choose a model m with data generating distribution $p(x\|\theta,m)$ and parameter prior $p(\theta\|m)$	Choose a model m with same data generating distribution $p(x\|\theta,m)$. No need for priors.
2. **Learning**	use Bayes rule to find the parameter posterior, $p(\theta\|D) =\propto p(D\|\theta)p(\theta)$	By Maximum Likelihood (ML) optimization, $\hat{\theta} = \arg\max_{\theta} p(D\|\theta)$
3. **Prediction**	$p(x\|D) = \int p(x\|\theta)p(\theta\|D)\,\mathrm{d}\theta$	$p(x\|D) = p(x\|\hat{\theta})$

Report Card on Maximum Likelihood Estimation

- Maximum Likelihood (ML) is MAP with uniform prior, or MAP is 'penalized' ML

$$\hat{\theta}_{map} = \arg\max_{\theta}\{ \overbrace{\log p(D|\theta)}^{\text{log-likelihood}} + \overbrace{\log p(\theta)}^{\text{penalty}}\}$$

- (good!). Works rather well if we have a lot of data because the influence of the prior diminishes with more data.

- (bad). Cannot be used for model comparison. E.g. best model does generally not correspond to largest likelihood (see part-2, Tjalkens).

- (good). Computationally often do-able. Useful fact (since $\log$ is monotonously increasing):

$$\arg\max_{\theta} \log p(D|\theta) = \arg\max_{\theta} p(D|\theta)$$

$\Rightarrow$ **ML estimation is an approximation to Bayesian learning**, but for good reason a very popular learning method when faced with lots of available data.

Working with Gaussians

Preliminaries

- Goal
  - Review of processing of Gaussian distributions in linear systems
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 85-93
    - MacKay - 2006 - The Humble Gaussian Distribution (highly recommended!)

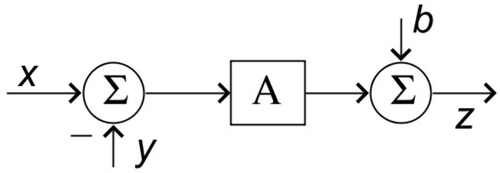
Sums and Transformations of Gaussian Variables

- The Gaussian distribution
$$\mathcal{N}(x|\mu,\Sigma) = |2\pi\Sigma|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}$$
for variable x is completely specified by its mean μ and variance Σ.
  - $\Lambda = \Sigma^{-1}$ is called the **precision matrix**.

- A **linear transformation** $z = Ax + b$ of a Gaussian variable $\mathcal{N}(x|\mu,\Sigma)$ is Gaussian distributed as
$$p(z) = \mathcal{N}\left(z \mid A\mu + b, A\Sigma A^T\right) \tag{SRG-4a}$$

- The **sum of two independent Gaussian variables** is also Gaussian distributed. Specifically, if $x \sim \mathcal{N}(x|\mu_x,\Sigma_x)$ and $y \sim \mathcal{N}(y|\mu_y,\Sigma_y)$, then the PDF for $z = x + y$ is given by
$$\begin{aligned} p(z) &= \mathcal{N}(x \mid \mu_x, \Sigma_x) \ast \mathcal{N}(y \mid \mu_y, \Sigma_y) \\ &= \mathcal{N}(z \mid \mu_x + \mu_y, \Sigma_x + \Sigma_y) \end{aligned} \tag{SRG-8}$$
  - [Exercise]: Show that Eq.SRG-8 is really a special case of Eq.SRG-4a.
  - The sum of two Gaussian *distributions* is NOT a Gaussian distribution. Why not?

Example: Gaussian Signals in a Linear System



- [Q.]: Given independent variables $x \sim \mathcal{N}(\mu_x,\sigma_y)$ and $y \sim \mathcal{N}(\mu_y,\sigma_y)$, what is the PDF for $z = A\cdot(x-y)+b$?

- [A.]: z is also Gaussian with
$$p_z(z) = \mathcal{N}(z|A(\mu_x-\mu_y)+b,\ A(\sigma_x+\sigma_y)A^T)$$

- Think about the role of the Gaussian distribution for stochastic linear systems in relation to what sinusoidals mean for deterministic linear system analysis.

Example: Bayesian Estimation of a Constant

- [Question] Estimate a constant θ from one 'noisy' measurement x about that constant. Assume the following model specification (the tilde $\sim$ means: 'is distributed as'):

$$\begin{aligned} x &= \theta + \epsilon \\ \theta &\sim \mathcal{N}(\mu_\theta, \sigma_\theta^2) \\ \epsilon &\sim \mathcal{N}(0, \sigma_\epsilon^2) \end{aligned}$$

[Answer]

- **1. Model specification** Note that you can rewrite these specifications in probabilistic notation as follows:

$$\begin{aligned} p(x|\theta) &= \mathcal{N}(x|\theta, \sigma_\epsilon^2) && \text{(likelihood)} \\ p(\theta) &= \mathcal{N}(\theta|\mu_\theta, \sigma_\theta^2) && \text{(prior)} \end{aligned}$$

- **2. Inference** for the posterior PDF $p(\theta|x)$

$$\begin{aligned} p(\theta|x) &= \frac{p(x|\theta)p(\theta)}{p(x)} = \frac{p(x|\theta)p(\theta)}{\int p(x|\theta)p(\theta)\,\mathrm{d}\theta} \\ &= \frac{1}{C}\,\mathcal{N}(x|\theta, \sigma_\epsilon^2)\,\mathcal{N}(\theta|\mu_\theta, \sigma_\theta^2) \\ &= \frac{1}{C_1}\exp\left\{-\frac{(x-\theta)^2}{2\sigma_\epsilon^2} - \frac{(\theta-\mu_\theta)^2}{2\sigma_\theta^2}\right\} \\ &= \frac{1}{C_1}\exp\left\{\theta^2\left(-\frac{1}{2\sigma_\epsilon^2} - \frac{1}{2\sigma_\theta^2}\right) + \theta\left(\frac{x}{\sigma_\epsilon^2} + \frac{\mu_\theta}{\sigma_\theta^2}\right) + C_2\right\} \\ &= \frac{1}{C_1}\exp\left\{-\frac{\sigma_\theta^2 + \sigma_\epsilon^2}{2\sigma_\theta^2\sigma_\epsilon^2}\left(\theta - \frac{x\sigma_\theta^2 + \mu_s\sigma_\epsilon^2}{\sigma_\theta^2 + \sigma_\epsilon^2}\right)^2 + C_3\right\} \end{aligned}$$

  which we recognize as a Gaussian distribution.

- This computational 'trick' for multiplying two Gaussians is called **completing the square**. The procedure makes use of the equality

$$ax^2 + bx + c_1 = a\left(x + \frac{b}{2a}\right)^2 + c_2$$

- Hence, it follows that the posterior for θ is

$$p(\theta|x) = \mathcal{N}(\theta|\,\mu_{\theta|x}, \sigma_{\theta|x}^2)$$

where

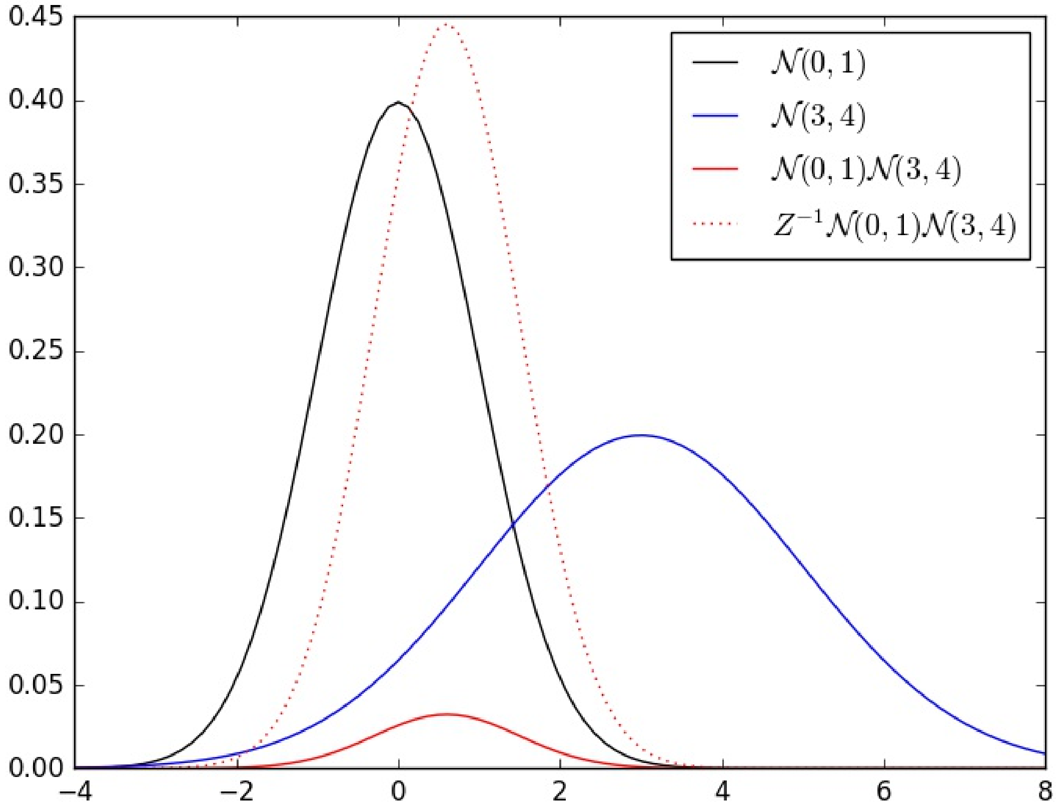
$$\begin{aligned} \frac{1}{\sigma_{\theta|x}^2} &= \frac{\sigma_\epsilon^2 + \sigma_\theta^2}{\sigma_\epsilon^2\sigma_\theta^2} = \frac{1}{\sigma_\theta^2} + \frac{1}{\sigma_\epsilon^2} \\ \mu_{\theta|x} &= \sigma_{\theta|x}^2\left(\frac{1}{\sigma_\epsilon^2}x + \frac{1}{\sigma_\theta^2}\mu_\theta\right) \end{aligned}$$

- So, multiplication of two Gaussian distributions yields another (unnormalized) Gaussian with
  - posterior precision equals **sum of prior precsions**
  - posterior precision-weighted mean equals **sum of prior precision-weighted means**

CODE EXAMPLE

Let's plot the exact product of two Gaussian PDFs as well as the normalized product according to the above derivation.

```
using PyPlot, Distributions
d1 = Normal(0, 1) # μ=0, σ^2=1
d2 = Normal(3, 2) # μ=3, σ^2=4

# Calculate the parameters of the product d1*d2
s2_prod = (d1.σ^-2 + d2.σ^-2)^-1
m_prod = s2_prod * ((d1.σ^-2)*d1.μ + (d2.σ^-2)*d2.μ)
d_prod = Normal(m_prod, sqrt(s2_prod)) # Note that we neglect the normalization constant.

# Plot stuff
x = linspace(-4, 8, 100)
plot(x, pdf.(d1,x), "k")
plot(x, pdf.(d2,x), "b")
plot(x, pdf.(d1,x) .* pdf.(d2,x), "r-") # Plot the exact product
plot(x, pdf.(d_prod,x), "r:")          # Plot the normalized Gaussian product
legend([L"\mathcal{N}(0,1)",
        L"\mathcal{N}(3,4)",
        L"\mathcal{N}(0,1) \mathcal{N}(3,4)",
        L"Z^{-1} \mathcal{N}(0,1) \mathcal{N}(3,4)"]);
```



The solid and dotted red curves are identical up to a scaling factor Z.

Multivariate Gaussian Multiplication

- In general, the multiplication of two multi-variate Gaussians yields an (unnormalized) Gaussian, see [SRG-6]:
$$\mathcal{N}(x|\mu_a,\Sigma_a)\cdot\mathcal{N}(x|\mu_b,\Sigma_b) = \alpha\cdot\mathcal{N}(x|\mu_c,\Sigma_c)$$
where
$$\Sigma_c^{-1} = \Sigma_a^{-1} + \Sigma_b^{-1}$$
$$\Sigma_c^{-1}\mu_c = \Sigma_a^{-1}\mu_a + \Sigma_b^{-1}\mu_b$$
and normalization constant $\alpha = \mathcal{N}(\mu_a|\,\mu_b,\Sigma_a+\Sigma_b)$.

- If we define the **precision** as $\Lambda \equiv \Sigma^{-1}$, then we see that **precisions add** and **precision-weighted means add** too.

- As we just saw, great application to Bayesian inference!

$$\underbrace{\text{Gaussian}}_{\text{posterior}} \propto \underbrace{\text{Gaussian}}_{\text{likelihood}} \times \underbrace{\text{Gaussian}}_{\text{prior}}$$

Conditioning and Marginalization of a Gaussian

- Let $z = \begin{bmatrix} x \\ y \end{bmatrix}$ be jointly normal distributed as

$$p(z) \quad = \mathcal{N}(z|\mu, \Sigma) = \mathcal{N}\left(\begin{bmatrix} x \\ y \end{bmatrix} \middle| \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_x & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_y \end{bmatrix} \right)$$

- Since covariance matrices are by definition symmetric, it follows that Σ_x and Σ_y are symmetric and $\Sigma_{xy} = \Sigma_{yx}^T$.

- Let's factorize $p(x, y)$ into $p(y|x)\, p(x)$ through conditioning and marginalization (proof in Bishop pp.87-89)

- **Conditioning**

$$\begin{aligned} p(y|x) &= p(x,y)/p(x) \\ &= \mathcal{N}\left(y|\mu_y + \Sigma_{yx}\Sigma_x^{-1}(x - \mu_x),\, \Sigma_y - \Sigma_{yx}\Sigma_x^{-1}\Sigma_{xy}\right) \end{aligned}$$

- **Marginalization**

$$p(x) = \int p(x,y)\,\mathrm{d}y = \mathcal{N}\left(x|\mu_x, \Sigma_x\right)$$

- Useful for applications to Bayesian inference in jointly Gaussian systems.

CODE EXAMPLE

Interactive plot of the joint, marginal, and conditional distributions.

```
using Reactive, Interact, PyPlot, Distributions
# z = [x; y]
μ = [1.; 2.]
Σ = [0.3 0.7;
     0.7 2.0]
joint = MvNormal(μ,Σ)
marginal_x = Normal(μ[1], sqrt(Σ[1,1]))

# Plot p(x,y)
subplot(221)
joint_pdf = Matrix(100,100)
x_range = linspace(-2,5,100); y_range = linspace(-2,5,100)
for i=1:length(x_range)
    for j=1:length(y_range)
        joint_pdf[i,j] = pdf(joint, [x_range[i];y_range[j]])
    end
end

imshow(joint_pdf', origin="lower", extent=[x_range[1], x_range[end], y_range[1], y_range[end]])
grid(); xlabel("x"); ylabel("y"); title("p(x,y)"); tight_layout()

# Plot p(x)
subplot(222)
plot(linspace(-2,5,100), pdf.(marginal_x, linspace(-2,5,100)))
grid(); xlabel("x"); ylabel("p(x)"); title("Marginal distribution p(x)"); tight_layout()

f = figure()
@manipulate for x=-2:0.1:3; withfig(f) do
        conditional_y_m = μ[2]+Σ[2,1]*inv(Σ[1,1])*(x-μ[1])
        conditional_y_s2 = Σ[2,2] - Σ[2,1]*inv(Σ[1,1])*Σ[1,2]
        conditional_y = Normal(conditional_y_m, sqrt.(conditional_y_s2))

        # Plot p(y|x)
        subplot(223)
        plot(linspace(-2,5,100), pdf.(conditional_y, linspace(-2,5,100)))
        grid(); xlabel("y"); ylabel("p(y|x)"); title("Conditional distribution p(y|x)"); tight_lay
out()
    end
end
```

As is clear from the plots, the conditional distribution is a renormalized slice from the joint distribution.

Example: Conditioning of Gaussian

- Consider (again) the system

$$\begin{aligned} x &= \theta + \epsilon \\ \theta &\sim \mathcal{N}(\theta|\mu_\theta,\sigma_\theta^2) \\ \epsilon &\sim \mathcal{N}(\epsilon|0,\sigma_\epsilon^2) \end{aligned}$$

- This system is equivalent to (derive this!)

$$p(x,\theta|\,\mu,\sigma) = \mathcal{N}\left(\begin{bmatrix} x\\ \theta \end{bmatrix} \,\middle|\, \begin{bmatrix} \mu_\theta\\ \mu_\theta \end{bmatrix}, \begin{bmatrix} \sigma_\theta^2 + \sigma_\epsilon^2 & \sigma_\theta^2\\ \sigma_\theta^2 & \sigma_\theta^2 \end{bmatrix} \right)$$

- Direct substitution of the rule for Gaussian conditioning leads to (derive this yourself!)

$$\begin{aligned} p(\theta|x) &= \mathcal{N}\left(\theta \,|\, \mu_{\theta|x}, \sigma_{\theta|x}^2\right), \quad \text{with} \\ K &= \frac{\sigma_\theta^2}{\sigma_\theta^2 + \sigma_\epsilon^2} \qquad (K \text{ is called: Kalman gain}) \\ \mu_{\theta|x} &= \mu_\theta + K\cdot(x-\mu_\theta) \\ \sigma_{\theta|x}^2 &= (1-K)\,\sigma_\theta^2 \end{aligned}$$

$\longrightarrow$ Moral: For jointly Gaussian systems, we can do inference simply in one step by using the formulas for conditioning and marginalization.

Application: Recursive Bayesian Estimation

Now consider the signal $x_t = \theta + \epsilon_t$, where $D_t = \{x_1, \ldots, x_t\}$ is observed *sequentially* (over time).

[Question]

- Derive a recursive algorithm for $p(\theta|D_t)$, i.e., an update rule for (posterior) $p(\theta|D_t)$ based on (prior) $p(\theta|D_{t-1})$ and (new observation) x_t.

[Answer]

- Let's define the estimate after t observations (i.e., our solution) as $p(\theta|D_t) = \mathcal{N}(\theta \mid \mu_t, \sigma_t^2)$.

- **Model specification**. We define the joint distribution for θ and x_t, given background D_{t-1}, by
$$\begin{aligned} p(x_t, \theta \mid D_{t-1}) &= p(x_t|\theta)\, p(\theta|D_{t-1}) \\ &= \underbrace{\mathcal{N}(x_t \mid \theta, \sigma_\epsilon^2)}_{\text{likelihood}} \underbrace{\mathcal{N}(\theta \mid \mu_{t-1}, \sigma_{t-1}^2)}_{\text{prior}} \end{aligned}$$

- **Inference**. Use Bayes rule,
$$\begin{aligned} p(\theta|D_t) &\propto p(x_t|\theta)\, p(\theta|D_{t-1}) \\ &= \mathcal{N}(x_t|\theta, \sigma_\epsilon^2)\, \mathcal{N}(\theta \mid \mu_{t-1}, \sigma_{t-1}^2) \\ &= \mathcal{N}(\theta|x_t, \sigma_\epsilon^2)\, \mathcal{N}(\theta \mid \mu_{t-1}, \sigma_{t-1}^2) \\ &= \mathcal{N}(\theta|\mu_t, \sigma_t^2) \end{aligned}$$
  with
$$\begin{aligned} K_t &= \frac{\sigma_{t-1}^2}{\sigma_{t-1}^2 + \sigma_\epsilon^2} \qquad \text{(Kalman gain)} \\ \mu_t &= \mu_{t-1} + K_t \cdot (x_t - \mu_{t-1}) \\ \sigma_t^2 &= (1 - K_t)\sigma_{t-1}^2 \end{aligned}$$
  (as before, we used the formulas for conditioning in a multivariate Gaussian system).

- This linear *sequential* estimator of mean and variance in Gaussian observations is called a **Kalman Filter**.

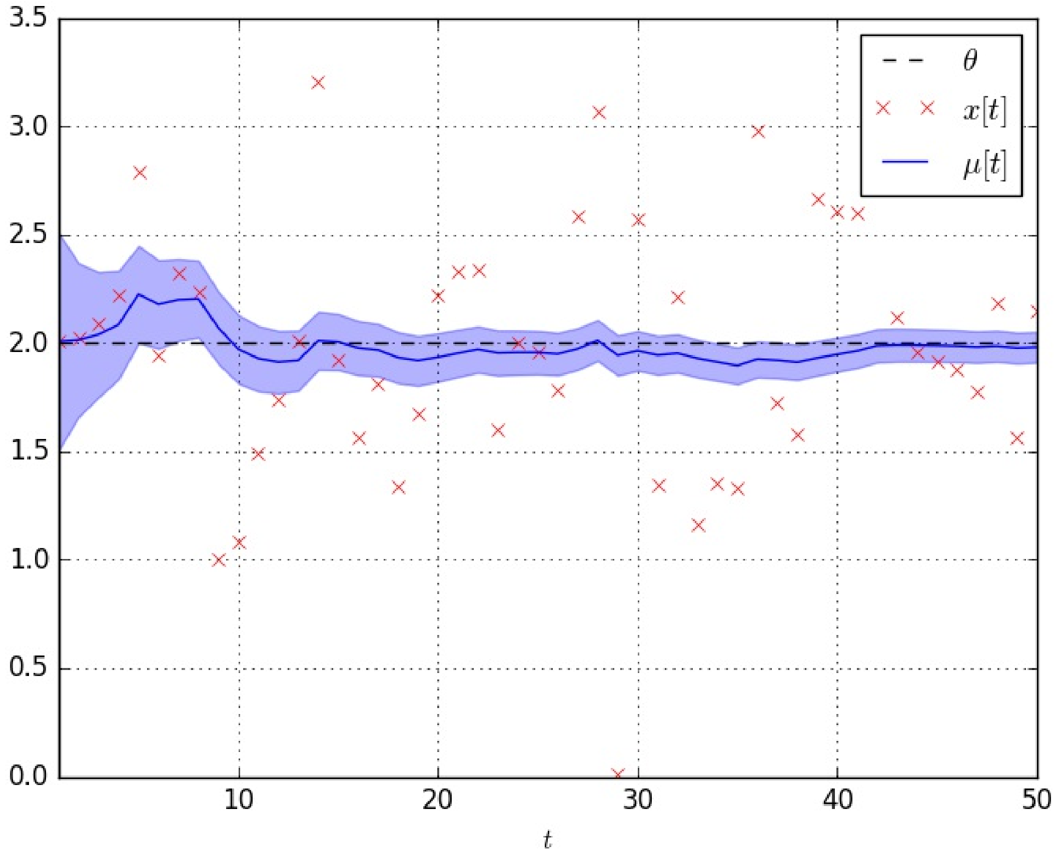
- Note that the uncertainty about θ decreases over time (since $0 < (1 - K_t) < 1$). This makes sense: since we assume that the statistics of the system do not change (stationarity), each new sample provides new information.

- Recursive Bayesian estimation is the basis for **adaptive signal processing** algorithms such as Least Mean Squares (LMS) and Recursive Least Squares (RLS).

CODE EXAMPLE

Let's implement the Kalman filter described above. We'll use it to recursively estimate the value of θ based on noisy observations. Use the 'Step' button to see the recursive updates to the posterior $p(\theta|D)$.

```
using PyPlot, Reactive, Interact

interactive_plot = false # Set to true to generate an interactive plot with 'step' button
N = 50                   # Number of observations
θ = 2.0                  # True value of the variable we want to estimate
σ_ϵ2 = 0.25              # Observation noise variance
x = sqrt(σ_ϵ2) * randn(N) + θ # Generate N noisy observations of θ

f = figure()
global t = 0
global μ = fill!(Vector{Float64}(N), NaN)    # Means of p(θ|D) over time
global σ_μ2 = fill!(Vector{Float64}(N), NaN) # Variances of p(θ|D) over time

function performKalmanStep()
    # Perform a Kalman filter step, update t, μ, σ_μ2
    global t += 1
    if t>1 # Use posterior from prev. step as prior
        K = σ_μ2[t-1] / (σ_ϵ2 + σ_μ2[t-1]) # Kalman gain
        μ[t] = μ[t-1] + K*(x[t] - μ[t-1])  # Update mean using (1)
        σ_μ2[t] = σ_μ2[t-1] * (1.0-K)      # Update variance using (2)
    elseif t==1 # Use prior
        # Prior p(θ) = N(0,1000)
        K = 1000.0 / (σ_ϵ2 + 1000.0) # Kalman gain
        μ[t] = 0 + K*(x[t] - 0)      # Update mean using (1)
        σ_μ2[t] = 1000 * (1.0-K)     # Update variance using (2)
    end
end

function plotStatus()
    # Plot the 'true' value of θ, noisy observations x, and the recursively updated posterior p(θ|
D)
    t = collect(1:N)
    plot(t, θ*ones(N), "k--")
    plot(t, x, "rx")
    plot(t, μ, "b-")
    fill_between(t, μ-sqrt.(σ_μ2), μ+sqrt.(σ_μ2), color="b", alpha=0.3)
    legend([L"\theta", L"x[t]", L"\mu[t]"])
    xlim((1, N)); xlabel(L"t"); grid()
end

if interactive_plot
    @manipulate for
        perform_step = button("Step");
        withfig(f) do
            if t<=N
                performKalmanStep()
                plotStatus()
            end
        end
    end
else
    while t<N
        performKalmanStep()
    end
    plotStatus()
end
```



The shaded area represents 2 standard deviations of posterior $p(\theta|D)$. The variance of the posterior is guaranteed to decrease monotonically for the standard Kalman filter.

Product of Normally Distributed Variables

- (We've seen that) the sum of two Gausssian distributed variables is also Gaussian distributed.

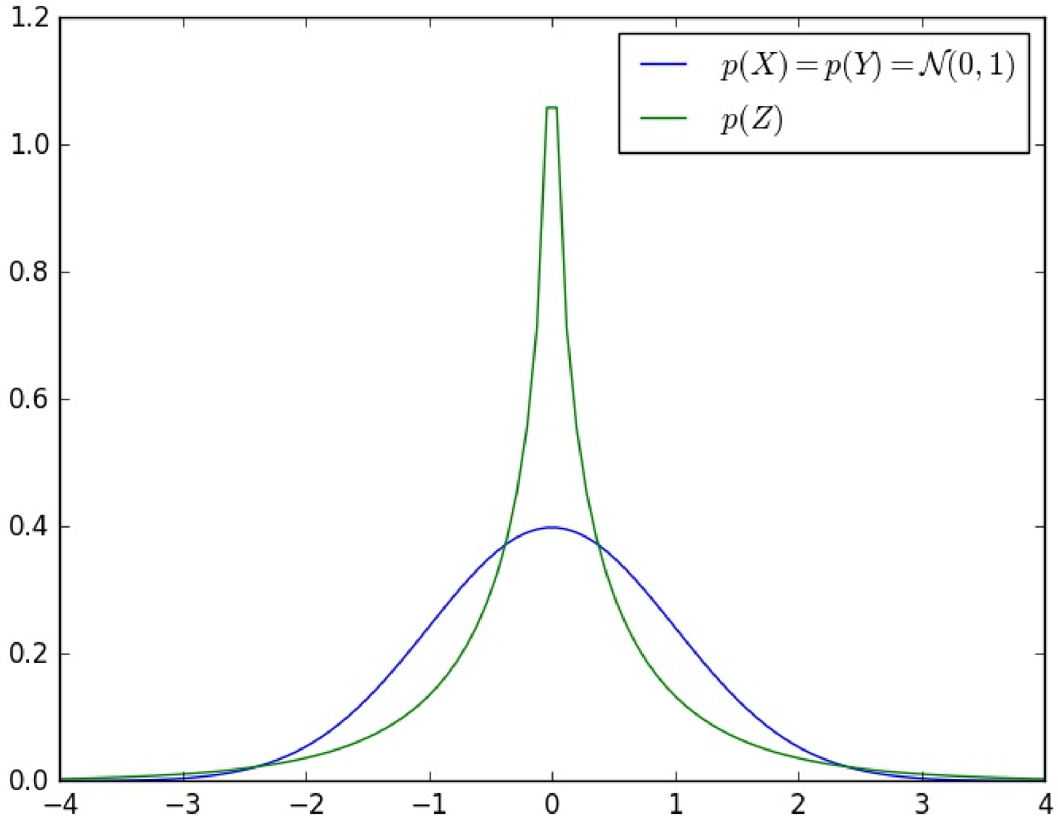
- Has the *product* of two Gaussian distributed variables also a Gaussian distribution?

- **No**! In general this is a difficult computation. As an example, let's compute $p(z)$ for $Z = XY$ for the special case that $X \sim \mathcal{N}(0,1)$ and $Y \sim \mathcal{N}(0,1)$.
$$
\begin{aligned}
p(z) &= \int_{X,Y} p(z|x,y)\,p(x,y)\,\mathrm{d}x\mathrm{d}y \\
&= \frac{1}{2\pi}\int \delta(z-xy)\,e^{-(x^2+y^2)/2}\,\mathrm{d}x\mathrm{d}y \\
&= \frac{1}{\pi}\int_0^\infty \frac{1}{x} e^{-(x^2+z^2/x^2)/2}\,\mathrm{d}x \\
&= \frac{1}{\pi}\mathrm{K}_0(|z|)\,.
\end{aligned}
$$
where $\mathrm{K}_n(z)$ is a modified Bessel function of the second kind.

CODE EXAMPLE

We plot $p(Z)$ to give an idea of what this distribution looks like.

```
using PyPlot, Distributions, SpecialFunctions
X = Normal(0,1)
pdf_product_std_normals(z::Vector) = (besselk.(0, abs.(z))./π)
range = collect(linspace(-4,4,100))
plot(range, pdf.(X, range))
plot(range, pdf_product_std_normals(range))
legend([L"p(X)=p(Y)=\mathcal{N}(0,1)", L"p(Z)"]);
```



Review Gaussians

The success of Gaussian distributions in probabilistic modeling is large due to the following properties:

- A linear transformation of a Gaussian distributed variable is also Gaussian distributed
- The convolution of two Gaussian functions is another Gaussian function (use in sum of 2 variables)
- The product of two Gaussian functions is another Gaussian function (use in Bayes rule).
- Conditioning and marginalization of multivariate Gaussian distributions produce Gaussians again (use in working with observations and when doing Bayesian predictions)
- The Gaussian PDF has higher entropy than any other with the same variance. (Not discussed in this course).
- Any smooth function with single rounded maximum, if raised to higher and higher powers, goes into a Gaussian function. (Not discussed).

Some Useful Matrix Calculus

Aside from working with Gaussians, it will be helpful for the next lessons to be familiar with some matrix calculus. We shortly recapitulate used formulas here.

- We define the **gradient** of a scalar function $f(A)$ w.r.t. an $n \times k$ matrix A as

$$\nabla_A f \triangleq \begin{bmatrix} \frac{\partial f}{\partial a_{11}} & \frac{\partial f}{\partial a_{12}} & \cdots & \frac{\partial f}{\partial a_{1k}} \\ \frac{\partial f}{\partial a_{21}} & \frac{\partial f}{\partial a_{22}} & \cdots & \frac{\partial f}{\partial a_{2k}} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial f}{\partial a_{n1}} & \frac{\partial f}{\partial a_{n2}} & \cdots & \frac{\partial f}{\partial a_{nk}} \end{bmatrix}$$

- The following formulas are useful (see Bishop App.-C)

$$\begin{aligned} |A^{-1}| &= |A|^{-1} && \text{(B-C.4)} \\ \nabla_A \log|A| &= (A^T)^{-1} = (A^{-1})^T && \text{(B-C.28)} \\ \mathrm{Tr}[ABC] &= \mathrm{Tr}[CAB] = \mathrm{Tr}[BCA] && \text{(B-C.9)} \\ \nabla_A \mathrm{Tr}[AB] &= \nabla_A \mathrm{Tr}[BA] = B^T && \text{(B-C.25)} \\ \nabla_A \mathrm{Tr}[ABA^T] &= A(B+B^T) && \text{(B-C.27)} \\ \nabla_x x^T A x &= (A+A^T)x && \text{(from B-C.27)} \\ \nabla_X a^T X b &= \nabla_X \mathrm{Tr}[ba^T X] = ab^T && \end{aligned}$$

What's Next?

- We discussed how Bayesian probability theory provides an integrated framework for making predictions based on observed data.

- The process involves model specification (your main task!), inference and actual model-based prediction.

- The latter two tasks are only difficult because of computational issues.

- Maximum likelihood was introduced as a computationally simpler approximation to the Bayesian approach.

- In particular under some linear Gaussian assumptions, a few interesting models can be designed.

- The rest of this course (part-1) concerns introduction to these Linear Gaussian models.

Density Estimation

Preliminaries

- Goal
  - Simple maximum likelihood estimates for Gaussian and categorical distributions
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 67-70, 74-76, 93-94

Why Density Estimation?

Density estimation relates to building a model $p(x|\theta)$ from observations $D=\{x_1,\dots,x_N\}$.

Why is this interesting? Some examples:

- **Outlier detection**. Suppose $D=\{x_n\}$ are benign mammogram images. Build $p(x|\theta)$ from D. Then low value for $p(x'|\theta)$ indicates that x' is a risky mammogram.

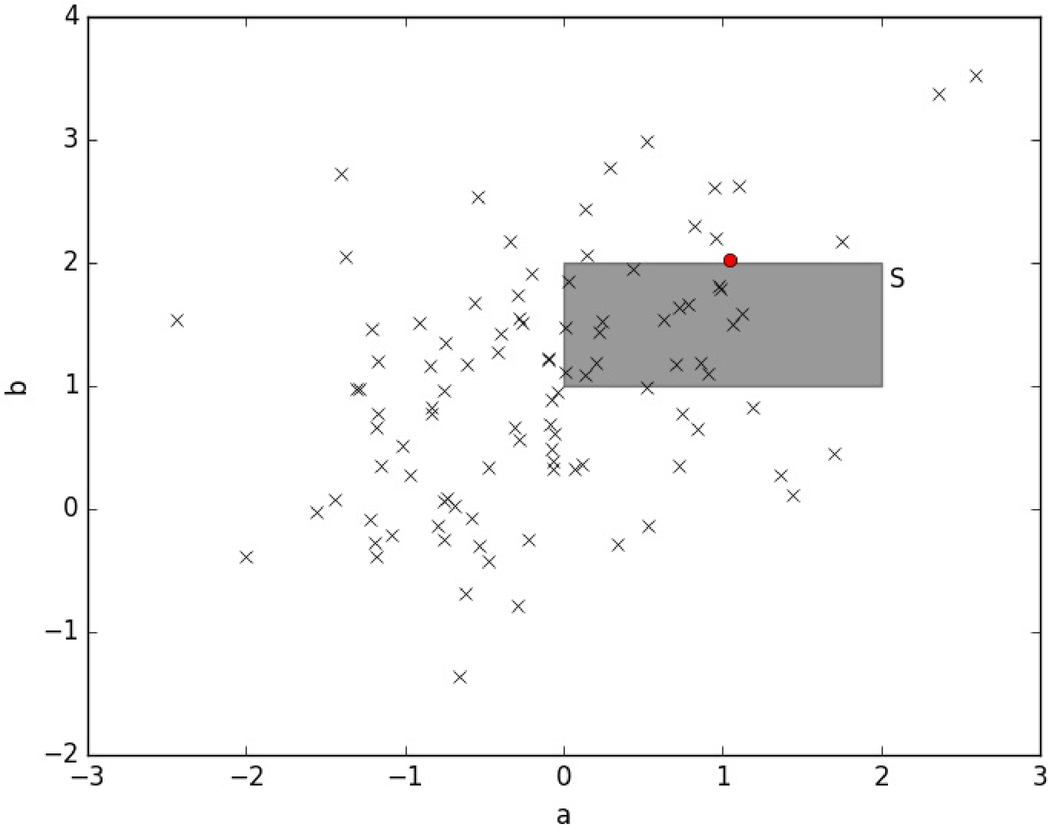
- **Compression**. Code a new data item based on **entropy**, which is a functional of $p(x|\theta)$:
$$H[p] = -\sum_x p(x|\theta)\log p(x|\theta)$$

- **Classification**. Let $p(x|\theta_1)$ be a model of attributes x for credit-card holders that paid on time and $p(x|\theta_2)$ for clients that defaulted on payments. Then, assign a potential new client x' to either class based on the relative probability of $p(x'|\theta_1)$ vs. $p(x'|\theta_2)$.

Example Problem

Consider a set of observations $D=\{x_1,\dots,x_N\}$ in the 2-dimensional plane (see Figure). All observations were generated by the same process. We now draw an extra observation $x_\bullet = (a,b)$ from the same data generating process. What is the probability that $x_\bullet$ lies within the shaded rectangle S?

```julia
using Distributions, PyPlot
N = 100
generative_dist = MvNormal([0,1.], [0.8 0.5; 0.5 1.0])
function plotObservations(obs::Matrix)
    plot(obs[1,:]', obs[2,:]', "kx", zorder=3)
    fill_between([0., 2.], 1., 2., color="k", alpha=0.4, zorder=2) # Shaded area
    text(2.05, 1.8, "S", fontsize=12)
    xlim([-3,3]); ylim([-2,4]); xlabel("a"); ylabel("b")
end
D = rand(generative_dist, N) # Generate observations from generative_dist
plotObservations(D)
x_dot = rand(generative_dist) # Generate x•
plot(x_dot[1], x_dot[2], "ro");
```



Log-Likelihood for a Multivariate Gaussian (MVG)

- Assume we are given a set of IID data points $D = \{x_1, \ldots, x_N\}$, where $x_n \in \mathbb{R}^D$. We want to build a model for these data.

- **Model specification**: Let's assume a MVG model $x_n = \mu + \epsilon_n$ with $\epsilon_n \sim \mathcal{N}(0, \Sigma)$, or equivalently,

$$p(x_n|\mu, \Sigma) \;= \mathcal{N}(x_n|\mu, \Sigma) = |2\pi\Sigma|^{-1/2} \exp\left\{-\frac{1}{2}(x_n - \mu)^T \Sigma^{-1} (x_n - \mu)\right\}$$

- Since the data are IID, $p(D|\theta)$ factorizes as

$$p(D|\theta) = p(x_1, \ldots, x_N|\theta) \stackrel{\text{IID}}{=} \prod_n p(x_n|\theta)$$

- This choice of model yields the following log-likelihood (use (B-C.9) and (B-C.4)),

$$\begin{aligned} \log p(D|\theta) &= \log \prod_n p(x_n|\theta) = \sum_n \log \mathcal{N}(x_n|\mu, \Sigma) \qquad (1) \\ &= N \cdot \log |2\pi\Sigma|^{-1/2} - \frac{1}{2} \sum_n (x_n - \mu)^T \Sigma^{-1} (x_n - \mu) \end{aligned}$$

Maximum Likelihood estimation of mean of MVG

- We want to maximize $\log p(D|\theta)$ wrt the parameters $\theta = \{\mu, \Sigma\}$. Let's take derivatives; first to mean μ, (making use of (B-C.25) and (B-C.27)),

$$\begin{aligned}\nabla_\mu \log p(D|\theta) &= -\frac{1}{2}\sum_n \nabla_\mu \left[(x_n-\mu)^T\Sigma^{-1}(x_n-\mu)\right] \\ &= -\frac{1}{2}\sum_n \nabla_\mu \mathbf{Tr}\left[-2\mu^T\Sigma^{-1}x_n + \mu^T\Sigma^{-1}\mu\right] \\ &= -\frac{1}{2}\sum_n \left(-2\Sigma^{-1}x_n + 2\Sigma^{-1}\mu\right) \\ &= \Sigma^{-1}\sum_n (x_n-\mu)\end{aligned}$$

- Setting the derivative to zero yields the **sample mean**

$$\hat{\mu} = \frac{1}{N}\sum_n x_n$$

Maximum Likelihood estimation of variance of MVG

- Now we take the gradient of the log-likelihood wrt the **precision matrix** Σ^{-1} (making use of B-C.28 and B-C.24)

$$\begin{aligned}&\nabla_{\Sigma^{-1}} \log p(D|\theta) \\ &= \nabla_{\Sigma^{-1}}\left[\frac{N}{2}\log|2\pi\Sigma|^{-1} - \frac{1}{2}\sum_{n=1}^N (x_n-\mu)^T\Sigma^{-1}(x_n-\mu)\right] \\ &= \nabla_{\Sigma^{-1}}\left[\frac{N}{2}\log|\Sigma^{-1}| - \frac{1}{2}\sum_{n=1}^N \mathbf{Tr}\left[(x_n-\mu)(x_n-\mu)^T\Sigma^{-1}\right]\right] \\ &= \frac{N}{2}\Sigma - \frac{1}{2}\sum_n (x_n-\mu)(x_n-\mu)^T\end{aligned}$$

Get optimum by setting the gradient to zero,

$$\hat{\Sigma} = \frac{1}{N}\sum_n (x_n-\hat{\mu})(x_n-\hat{\mu})^T$$

which is also known as the **sample variance**.

Sufficient Statistics

- Note that the ML estimates can also be written as

$$\hat{\Sigma} = \sum_n x_n x_n^T - \left(\sum_n x_n\right)\left(\sum_n x_n\right)^T, \quad \hat{\mu} = \frac{1}{N}\sum_n x_n$$

- I.o.w., the two statistics (a 'statistic' is a function of the data) $\sum_n x_n$ and $\sum_n x_n x_n^T$ are sufficient to estimate the parameters μ and Σ from N observations. In the literature, $\sum_n x_n$ and $\sum_n x_n x_n^T$ are called **sufficient statistics** for the Gaussian PDF.

- The actual parametrization of a PDF is always a re-parameteriation of the sufficient statistics.

- Sufficient statistics are useful because they summarize all there is to learn about the data set in a minimal set of variables.

Solution to Example Problem

We apply maximum likelihood estimation to fit a 2-dimensional Gaussian model (m) to data set D. Next, we evaluate $p(x_\bullet \in S|m)$ by (numerical) integration of the Gaussian pdf over S: $p(x_\bullet \in S|m) = \int_S p(x|m)\mathrm{d}x$.

```
using Cubature # Numerical integration package

# Maximum likelihood estimation of 2D Gaussian
μ = 1/N * sum(D,2)[:,1]
D_min_μ = D - repmat(μ, 1, N)
Σ = Hermitian(1/N * D_min_μ*D_min_μ')
m = MvNormal(μ, convert(Matrix, Σ))

# Contour plot of estimated Gaussian density
A = Matrix{Float64}(100,100); B = Matrix{Float64}(100,100)
density = Matrix{Float64}(100,100)
for i=1:100
    for j=1:100
        A[i,j] = a = (i-1)*6/100.-2
        B[i,j] = b = (j-1)*6/100.-3
        density[i,j] = pdf(m, [a,b])
    end
end
c = contour(A, B, density, 6, zorder=1)
PyPlot.set_cmap("cool")
clabel(c, inline=1, fontsize=10)

# Plot observations, x•, and the countours of the estimated Gausian density
plotObservations(D)
plot(x_dot[1], x_dot[2], "ro")

# Numerical integration of p(x|m) over S:
(val,err) = hcubature((x)->pdf(m,x), [0., 1.], [2., 2.])
println("p(x·∈S|m) ≈ $(val)")
```

Discrete Data: the 1-of-K Coding Scheme

- Consider a coin-tossing experiment with outcomes $x \in \{0,1\}$ (tail and head) and let $0 \leq \mu \leq 1$ represent the probability of heads. This model can written as a **Bernoulli distribution**:

$$p(x|\mu) = \mu^x (1-\mu)^{1-x}$$

  - Note that the variable x acts as a (binary) **selector** for the tail or head probabilities. Think of this as an 'if'-statement in programming.

- **1-of-K coding scheme**. Now consider a K -sided coin (a *die* (pl.: dice)). It is convenient to code the outcomes by $x = (x_1, \ldots, x_K)^T$ with **binary selection variables**

$$x_k = \begin{cases} 1 & \text{if die landed on } k\text{th face} \\ 0 & \text{otherwise} \end{cases}$$

  - E.g., for $K = 6$, if the die lands on the 3rd face $\Rightarrow x = (0,0,1,0,0,0)^T$.

- Assume the probabilities $p(x_k = 1) = \mu_k$ with $\sum_k \mu_k = 1$. The data generating distribution is then (note the similarity to the Bernoulli distribution)

$$p(x|\mu) = \mu_1^{x_1} \mu_2^{x_2} \cdots \mu_k^{x_k} = \prod_k \mu_k^{x_k}$$

- This generalized Bernoulli distribution is called the **categorical distribution** (or sometimes the 'multi-noulli' distribution).

Categorical vs. Multinomial Distribution

- Observe a data set $D=\{x_1,\ldots,x_N\}$ of N IID rolls of a K-sided die, with generating PDF
$$p(D|\mu) = \prod_n \prod_k \mu_k^{x_{nk}} = \prod_k \mu_k^{\sum_n x_{nk}} = \prod_k \mu_k^{m_k}$$
where $m_k = \sum_n x_{nk}$ is the total number of occurrences that we 'threw' k eyes.

- This distribution depends on the observations **only** through the quantities $\{m_k\}$, with generally $K \ll N$.

- A related distribution is the distribution over $D_m=\{m_1,\ldots,m_K\}$, which is called the **multinomial distribution**,
$$p(D_m|\mu) = \frac{N!}{m_1! m_2! \ldots m_K!} \prod_k \mu_k^{m_k}\,.$$

- The catagorical distribution $p(D|\mu) = p(\,x_1,\ldots,x_N\,|\,\mu\,)$ is a distribution over the **observations** $\{x_1,\ldots,x_N\}$, whereas the multinomial distribution $p(D_m|\mu) = p(\,m_1,\ldots,m_K\,|\,\mu\,)$ is a distribution over the **data frequencies** $\{m_1,\ldots,m_K\}$.

Maximum Likelihood Estimation for the Multinomial

- Now let's find the ML estimate for μ, based on N throws of a K-sided die. Again we use the shorthand $m_k \triangleq \sum_n x_{nk}$.

- The log-likelihood for the multinomial distribution is given by
$$\mathrm{L}(\mu) \triangleq \log p(D_m|\mu) \propto \log \prod_k \mu_k^{m_k} = \sum_k m_k \log \mu_k \tag{2}$$

- When doing ML estimation, we must obey the constraint $\sum_k \mu_k = 1$, which can be accomplished by a Lagrange multiplier. The **augmented log-likelihood** with Lagrange multiplier is then
$$\mathrm{L}'(\mu) = \sum_k m_k \log \mu_k + \lambda \cdot \big(1 - \sum_k \mu_k\big)$$

- Set derivative to zero yields the **sample proportion** for μ_k
$$\nabla_{\mu_k} \mathrm{L}' = \frac{m_k}{\hat\mu_k} - \lambda \overset{!}{=} 0 \;\Rightarrow\; \hat\mu_k = \frac{m_k}{N}$$

where we get λ from the constraint
$$\sum_k \hat\mu_k = \sum_k \frac{m_k}{\lambda} = \frac{N}{\lambda} \overset{!}{=} 1$$

Recap ML for Density Estimation

Given N IID observations $D=\{x_1,\ldots,x_N\}$.

- For a **multivariate Gaussian** model $p(x_n|\theta) = \mathcal{N}(x_n|\mu,\Sigma)$, we obtain ML estimates
$$\hat\mu = \frac{1}{N}\sum_n x_n \qquad \text{(sample mean)}$$
$$\hat\Sigma = \frac{1}{N}\sum_n (x_n-\hat\mu)(x_n-\hat\mu)^T \qquad \text{(sample variance)}$$

- For discrete outcomes modeled by a 1-of-K **categorical distribution** we find

$$\hat{\mu}_k = \frac{1}{N} \sum_n x_{nk} \quad \left(= \frac{m_k}{N} \right) \qquad \text{(sample proportion)}$$

- Note the similarity for the means between discrete and continuous data.

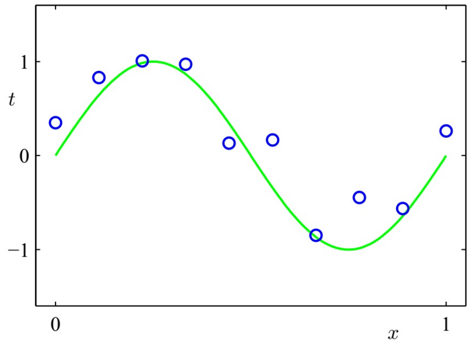
- We didn't use a co-variance matrix for discrete data. Why?

Linear Regression

Preliminaries

- Goal
  - Maximum likelihood estimates for various linear regression variants
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 140-144
    - G. Deng et al., *A model-based approach for the development of LMS algorithms*', ISCAS-05 symposium, 2005.

Regression - Illustration



Given a set of (noisy) data measurements, find the 'best' relation between an input variable $x \in \mathbb{R}^D$ and input-dependent outcomes $y \in \mathbb{R}$

Regression vs Density Estimation

- Observe N IID data **pairs** $D = \{(x_1, y_1), \dots, (x_N, y_N)\}$ with $x_n \in \mathbb{R}^D$ and $y_n \in \mathbb{R}$.

- [Q.] We could try to build a model for the data by density estimation, $p(x, y)$, but what if we are interested only in (a model for) the responses y_n for **given inputs** x_n ?

- [A.] We will build a model only for the conditional distribution $p(y|x)$.
  - Note that, since $p(x, y) = p(y|x)\, p(x)$, this is a building block for the joint data density.
  - In a sense, this is density modeling with the assumption that x is drawn from a uniform distribution.

- Next, we discuss model (1) specification, (2) ML estimation and (3) prediction for the linear regression model.

1. Model Specification for Linear Regression

- In a *regression* model, we try to 'explain the data' by a purely deterministic term $f(x,w)$, plus a purely random term ϵ_n for 'unexplained noise',

$$y_n = f(x_n, w) + \epsilon_n$$

- In *linear regression*, we assume that

$$f(x,w) = w^T x \,.$$

- In *ordinary linear regression*, the noise process ϵ_n is zero-mean Gaussian with constant variance σ^2, i.e.

$$y_n = w^T x_n + \mathcal{N}(0,\sigma^2) \,,$$

or equivalently, the likelihood model is

$$p(y_n|\, x_n, w) = \mathcal{N}(y_n|\, w^T x_n, \sigma^2) \,.$$

- For full Bayesian learning we should also choose a prior $p(w)$; In ML estimation, the prior $p(w)$ is uniformly distributed (so it can be ignored).

2. ML Estimation for Linear Regression Model

- Let's work out the log-likelihood for multiple observations

$$\begin{aligned}\log p(D|w) &\stackrel{\text{IID}}{=} \sum_n \log \mathcal{N}(y_n \,|\, w^T x_n, \sigma^2) \propto -\frac{1}{2\sigma^2} \sum_n (y_n - w^T x_n)^2 \\ &= -\frac{1}{2\sigma^2} (y - \mathbf{X}w)^T (y - \mathbf{X}w)\end{aligned}$$

  where we defined $N \times 1$ vector $y = (y_1, y_2, \dots, y_N)^T$ and $(N \times D)$-dim matrix $\mathbf{X} = (x_1, x_2, \dots, x_n)^T$.

- Set the derivative $\nabla_w \log p(D|w) = \frac{1}{\sigma^2} \mathbf{X}^T (y - \mathbf{X}w)$ to zero for the maximum likelihood estimate

$$\hat{w}_{\text{ML}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T y$$

- The matrix $\mathbf{X}^\dagger \equiv (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$ is also known as the **Moore-Penrose pseudo-inverse** (which is sort-of-an-inverse for non-square matrices).

- Note that size ($N \times D$) of the data matrix $\mathbf{X}$ grows with number of observations, but the size ($D \times D$) of $\mathbf{X}^T \mathbf{X}$ is independent of training data set.

3. Prediction of New Data Points

- Now, we want to apply the trained model. New data points can be predicted by

$$p(y_\bullet \mid x_\bullet, \hat{w}_{\text{ML}}) = \mathcal{N}(y_\bullet \mid \hat{w}^T_{\text{ML}} x_\bullet, \sigma^2)$$

- Note that the expected value of a predicted new data point

$$\mathrm{E}[y_\bullet] = \hat{w}^T_{\text{ML}} x_\bullet = x_\bullet^T \hat{w}_{\text{ML}} = \left(x_\bullet^T \mathbf{X}^\dagger\right) y$$

can also be expressed as a linear combination of the observed data points

$$y = (y_1, y_1, \dots, y_N)^T \,.$$

Deterministic Least-Squares Regression

- (You may say that) we don't need to work with probabilistic models. E.g., there's also the deterministic **least-squares** solution: minimize sum of squared errors,

$$\hat{w}_{\text{LS}} = \arg\min_{w} \sum_n \left(y_n - w^T x_n\right)^2 = \arg\min_{w} (y - \mathbf{X}w)^T (y - \mathbf{X}w)$$

- Setting the gradient $\frac{\partial (y-\mathbf{X}w)^T(y-\mathbf{X}w)}{\partial w} = -2\mathbf{X}^T(y-\mathbf{X}w)$ to zero yields the **normal equations** $\mathbf{X}^T\mathbf{X}\hat{w}_{\text{LS}} = \mathbf{X}^T y$ and consequently

$$\hat{w}_{\text{LS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y$$

which is the same answer as we got for the maximum likelihood weights $\hat{w}_{\text{ML}}$.

- $\Rightarrow$ Least-squares regression ($\hat{w}_{\text{LS}}$) corresponds to (probabilistic) maximum likelihood ($\hat{w}_{\text{ML}}$) if
  1. **IID samples** (determines how errors are combined), and
  2. Noise $\epsilon_n \sim \mathcal{N}(0,\,\sigma^2)$ is **Gaussian** (determines error metric)

Probabilistic vs. Deterministic Approach

- The (deterministic) least-squares approach assumed IID Gaussian distributed data, but these assumptions are not obvious from looking at the least-squares (LS) criterion.

- If the data were better modeled by non-Gaussian assumptions (or not IID), then LS might not be appropriate.

- The probabilistic approach makes all these issues completely transparent by focusing on the **model specification** rather than the error criterion.

- Next, we will show this by two examples: (1) samples not identically distributed, and (2) few data points.

Not Identically Distributed Data

- What if we assume that the variance of the measurement error varies with the sampling index, $\epsilon_n \sim \mathcal{N}(0,\sigma_n^2)$?

- Let's make the log-likelihood again (use $\Lambda \triangleq \text{diag}[1/\sigma_n^2]$):

$$\mathrm{L(w)} \triangleq \log p(D|w) \propto -\frac{1}{2}\sum_n \frac{(y_n - w^T x_n)^2}{\sigma_n^2} = -\frac{1}{2}(y-\mathbf{X}w)^T\Lambda(y-\mathbf{X}w)\,.$$

- Set derivative $\partial\mathrm{L(w)}/\partial w = -\mathbf{X}^T\Lambda(y-\mathbf{X}w)$ to zero to get the **normal equations** $\mathbf{X}^T\Lambda\mathbf{X}\hat{w}_{\text{WLS}} = \mathbf{X}^T\Lambda y$ and consequently

$$\hat{w}_{\text{WLS}} = \left(\mathbf{X}^T\Lambda\mathbf{X}\right)^{-1}\mathbf{X}^T\Lambda y$$

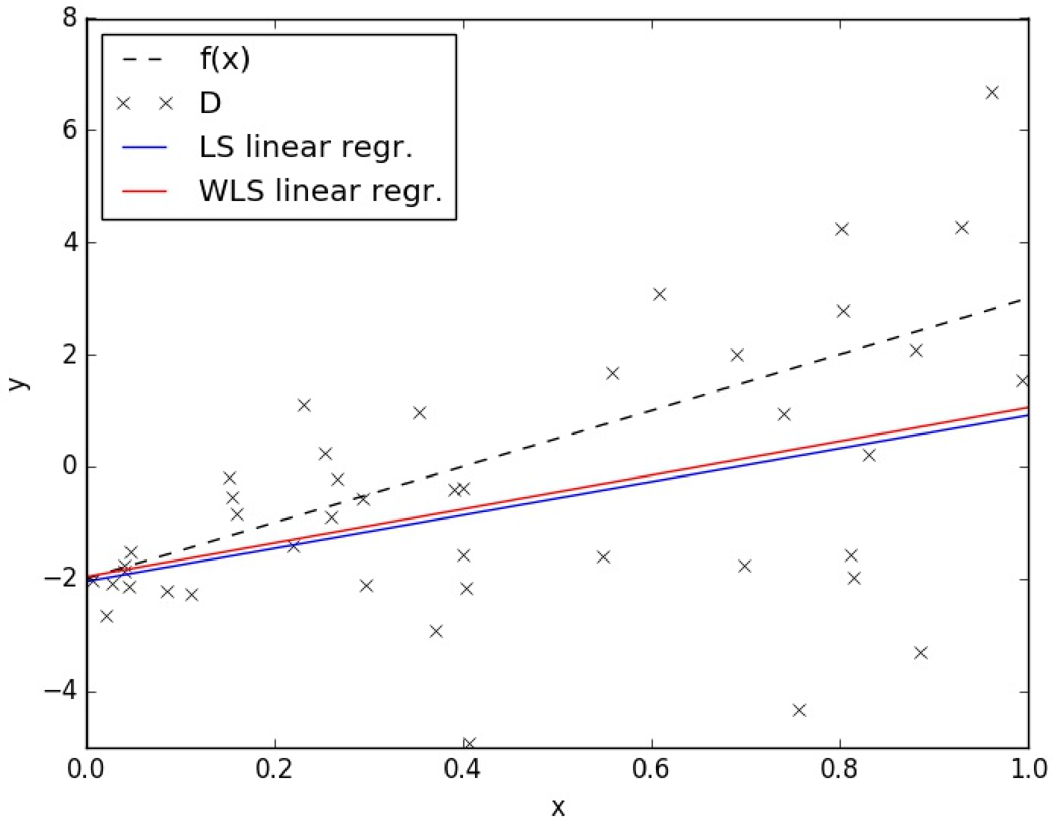
- This is also called the **Weighted Least Squares** (WLS) solution. (Note that we just stumbled upon it, the crucial aspect is appropriate model specification!)

- Note also that the dimension of Λ grows with the number of data points. In general, models for which the number of parameters grow as the number of observations increase are called **non-parametric models**.

We'll compare the Least Squares and Weighted Least Squares solutions for a simple linear regression model with input-dependent noise:

$$
\begin{aligned}
x &\sim \mathrm{Unif}[0,1] \\
y|x &\sim \mathcal{N}(f(x), v(x)) \\
f(x) &= 5x - 2 \\
v(x) &= 10e^{2x^2} - 9.5 \\
\mathcal{D} &= \{(x_1, y_1), \ldots, (x_N, y_N)\}
\end{aligned}
$$

```
using PyPlot

# Model specification: y|x ~ (f(x), v(x))
f(x) = 5*x - 2
v(x) = 10*exp.(2*x.^2) - 9.5 # input dependent noise variance
x_test = [0.0, 1.0]
plot(x_test, f(x_test), "k--") # plot f(x)

# Generate N samples (x,y), where x ~ Unif[0,1]
N = 50
x = rand(N)
y = f(x) + sqrt.(v(x)) .* randn(N)
plot(x, y, "kx"); xlabel("x"); ylabel("y") # Plot samples

# Add constant to input so we can estimate both the offset and the slope
_x = [x ones(N)]
_x_test = hcat(x_test, ones(2))

# LS regression
w_ls = pinv(_x) * y
plot(x_test, _x_test*w_ls, "b-") # plot LS solution

# Weighted LS regression
W = diagm(1./v(x)) # weight matrix
w_wls = inv(_x'*W*_x) * _x' * W * y
plot(x_test, _x_test*w_wls, "r-") # plot WLS solution
ylim([-5,8]); legend(["f(x)", "D", "LS linear regr.", "WLS linear regr."],loc=2);
```



Too Few Training Samples

- If we have fewer training samples than input dimensions, $\mathbf{X}^T\mathbf{X}$ will not be invertible. (Why?)

- As a general recipe, in case of (expected) problems, **go back to full Bayesian!** Do proper model specification, Bayesian inference etc. Let's do this next.

- **Model specification**. Let's try a Gaussian prior for w (why is this reasonable?)

$$p(w) = \mathcal{N}(w|0,\Sigma) = \mathcal{N}(w|0,\varepsilon I)$$

- **Learning**. Let's do Bayesian inference,

$$\begin{aligned}
\log p(w|D) &\propto \log p(D|w)p(w) \\
&\stackrel{IID}{=} \log \sum_n p(y_n|x_n,w) + \log p(w) \\
&= \log \sum_n \mathcal{N}(y_n|\,w^T x_n,\sigma^2) + \log \mathcal{N}(w|0,\varepsilon I) \\
&\propto \frac{1}{2\sigma^2}(y-\mathbf{X}w)^T(y-\mathbf{X}w) + \frac{1}{2\epsilon}w^T w
\end{aligned}$$

- **Done!** The posterior $p(w|D)$ specifies all we know about w after seeing the data.

Too Few Training Samples, cont'd: the MAP estimate

- As discussed, for practical purposes, you often want a point estimate for w, rather than a posterior distribution.

- For instance, let's take a **Maximum A Posteriori (MAP) estimate**. Set derivative

$$\nabla_w \log p(w|D) = -\frac{1}{\sigma^2}\mathbf{X}^T(y-\mathbf{X}w) + \frac{1}{\varepsilon}w$$

  to zero, yielding

$$\hat{w}_{\text{MAP}} = \left(\mathbf{X}^T\mathbf{X} + \frac{\sigma^2}{\varepsilon}I\right)^{-1}\mathbf{X}^T y$$

- Note that, in contrast to $\mathbf{X}^T\mathbf{X}$, the matrix $\left(\mathbf{X}^T\mathbf{X} + (\sigma^2/\varepsilon)I\right)$ is always invertible! (Why?)

- Note also that $\hat{w}_{\text{LS}}$ is retrieved by letting $\varepsilon \rightarrow \infty$. Does that make sense?

Adaptive Linear Regression

- What if the data arrives one point at a time?

- Two standard *adaptive* linear regression approaches: RLS and LMS. Here we shortly recap the LMS approach.

- **Least Mean Squares** (LMS) is gradient-descent on a 'local-in-time' approximation of the square-error cost function.

- Define the cost-of-current-sample as

$$E_n(w) = \frac{1}{2}(y_n - w^T x_n)^2$$

and track the optimum by gradient descent (at each sample index n):

$$w_{n+1} = w_n - \eta \left.\frac{\partial E_n}{\partial w}\right|_{w_n}$$

which leads to the LMS update:

$$w_{n+1} = w_n + \eta \, (y_n - w_n^T x_n) \, x_n$$

- (OPTIONAL) This is not a probabilistic modelling derivation. Is there also a Bayesian treatment of LMS? Sure, e.g., have a look at G. Deng et al., *A model-based approach for the development of LMS algorithms*', ISCAS-05 symposium, 2005.

Generative Classification

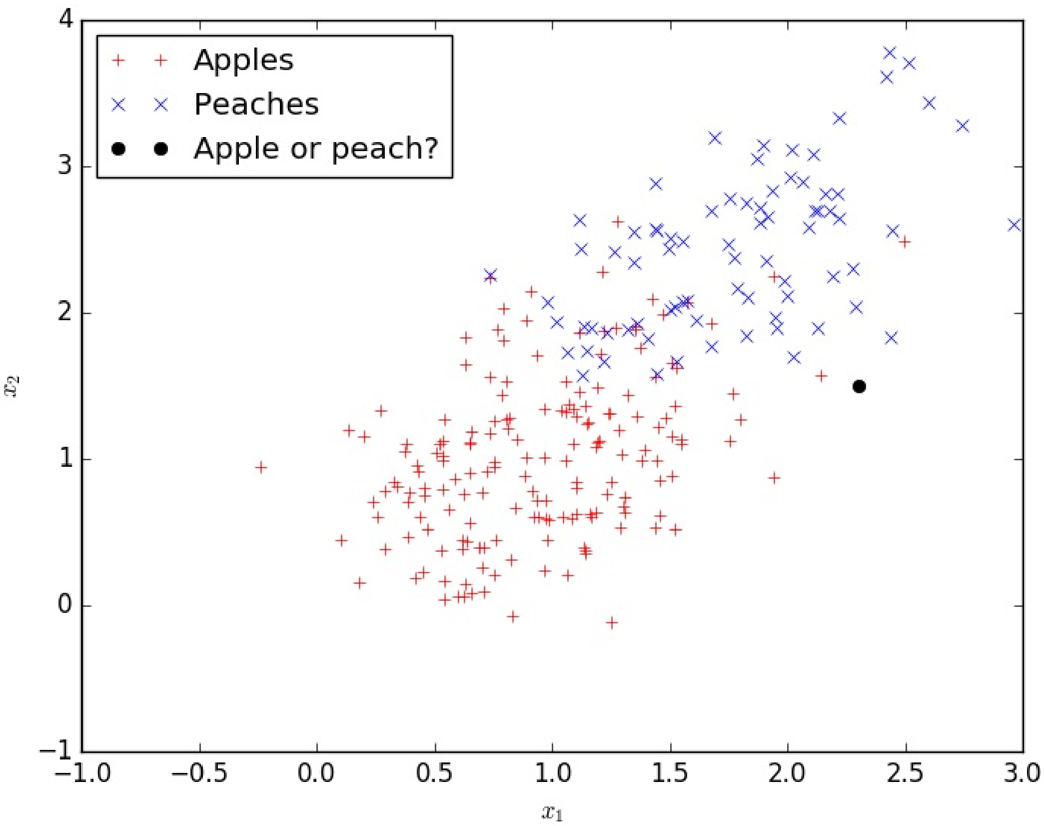
Preliminaries

- Goal
  - Introduction to linear generative classification with multinomial-Gaussian generative model
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 196-202

Example Problem: an apple or a peach?

You're given numerical values for the skin features roughness and color for 200 pieces of fruit, where for each piece of fruit you also know if it is an apple or a peach. Now you receive the roughness and color values for a new piece of fruit but you don't get its class label (apple or peach). What is the probability that the new piece is an apple?

```julia
using Distributions, PyPlot
N = 250; p_apple = 0.7; Σ = [0.2 0.1; 0.1 0.3]
p_given_apple = MvNormal([1.0, 1.0], Σ) # p(X|y=apple)
p_given_peach = MvNormal([1.7, 2.5], Σ) # p(X|y=peach)
X = Matrix{Float64}(2,N); y = Vector{Bool}(N) # true corresponds to apple
for n=1:N
    y[n] = (rand() < p_apple) # Apple or peach?
    X[:,n] = y[n] ? rand(p_given_apple) : rand(p_given_peach) # Sample features
end
X_apples = X[:,find(y)]'; X_peaches = X[:,find(.!y)]' # Sort features on class
x_test = [2.3; 1.5] # Features of 'new' data point

function plot_fruit_dataset()
    # Plot the data set and x_test
    plot(X_apples[:,1], X_apples[:,2], "r+")   # apples
    plot(X_peaches[:,1], X_peaches[:,2], "bx") # peaches
    plot(x_test[1], x_test[2], "ko")           # 'new' unlabelled data point
    legend(["Apples"; "Peaches"; "Apple or peach?"], loc=2)
    xlabel(L"x_1"); ylabel(L"x_2"); xlim([-1,3]); ylim([-1,4])
end
plot_fruit_dataset();
```



Generative Classification Problem Statement

- Given is a data set $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
  - inputs $x_n \in \mathbb{R}^D$ are called **features**.
  - outputs $y_n \in \mathcal{C}_k$, with $k = 1, \ldots, K$; The **discrete** targets $\mathcal{C}_k$ are called **classes**.

- We will again use the 1-of-K notation for the discrete classes. Define the binary **class selection variable**

$$y_{nk} = \begin{cases} 1 & \text{if } y_n \in \mathcal{C}_k \\ 0 & \text{otherwise} \end{cases}$$

  - (Hence, the notations $y_{nk} = 1$ and $y_n \in \mathcal{C}_k$ mean the same thing.)

- The plan for generative classification: build a model for the joint pdf $p(x,y)=p(x|y)p(y)$ and use Bayes to infer the posterior class probabilities

$$p(y|x) = \frac{p(x|y)p(y)}{\sum_y p(x|y)p(y)} \propto p(x|y)\,p(y)$$

1 – Model specification

Likelihood

- Assume Gaussian **class-conditional distributions** with **constant covariance matrix** across the classes,

$$p(x_n|\mathcal{C}_k) = \mathcal{N}(x_n|\mu_k,\Sigma)$$

  with notational shorthand: $\mathcal{C}_k \triangleq (y_n \in \mathcal{C}_k)$.

Prior

- We use a categorical distribution for the class labels y_{nk}:

$$p(\mathcal{C}_k) = \pi_k$$

- This leads to

$$p(x_n,\mathcal{C}_k) = \pi_k \cdot \mathcal{N}(x_n|\mu_k,\Sigma)$$

- The log-likelihood for the full data set is then

$$\begin{aligned}
\log\, p(D|\theta) &\stackrel{\text{IID}}{=} \sum_n \log p(x_n,\mathcal{C}_1,\ldots,\mathcal{C}_K \mid \theta) \\
&= \sum_n \log \prod_k p(x_n,\mathcal{C}_k|\theta)^{y_{nk}} \quad \text{(use 1-of-K coding)} \\
&= \sum_{n,k} y_{nk} \log p(x_n,\mathcal{C}_k|\theta) \\
&= \sum_{n,k} y_{nk} \log \mathcal{N}(x_n|\mu_k,\Sigma) + \sum_{n,k} y_{nk} \log \pi_k \\
&= \sum_{n,k} y_{nk} \underbrace{\log \mathcal{N}(x_n|\mu_k,\Sigma)}_{\text{see Gaussian est.}} + \underbrace{\sum_k m_k \log \pi_k}_{\text{see multinomial est.}}
\end{aligned}$$

  where we used $m_k \triangleq \sum_n y_{nk}$.

- As usual, the rest (inference for parameters and model prediction) through straight probability theory.

2 – Parameter Inference for Classification

- We'll do ML estimation for $\theta = \{\pi_k,\mu_k,\Sigma\}$ from data D

- Recall (from the previous slide) the log-likelihood (LLH)

$$\log\, p(D|\theta) = \sum_{n,k} y_{nk} \underbrace{\log \mathcal{N}(x_n|\mu_k,\Sigma)}_{\text{Gaussian}} + \underbrace{\sum_k m_k \log \pi_k}_{\text{multinomial}}$$

- Maximization of the LLH breaks down into
  - **Gaussian density estimation** for parameters μ_k,Σ, since the first term contains exactly the LLH for MVG density estimation (see lesson on Density Est., Eq.1)
  - **Multinomial density estimation** for class priors π_k, since the second term holds exactly the LLH for multinomial density estimation (see lesson on Density Estimation, Eq.2).

- The ML for multinomial class prior (we've done this before!)
$$\pi_k = m_k/N$$

- Now group the data into separate classes and do MVG ML estimation for class-conditional parameters (we've done this before as well):
$$\begin{aligned}
\hat{\mu}_k &= \frac{\sum_n y_{nk} x_n}{\sum_n y_{nk}} = \frac{1}{m_k} \sum_n y_{nk} x_n \\
\hat{\Sigma} &= \frac{1}{N} \sum_{n,k} y_{nk} (x_n - \hat{\mu}_k)(x_n - \hat{\mu}_k)^T \\
&= \sum_k \hat{\pi}_k \cdot \underbrace{\left(\frac{1}{m_k} \sum_n y_{nk} (x_n - \hat{\mu}_k)(x_n - \hat{\mu}_k)^T \right)}_{\text{class-cond. variance}} \\
&= \sum_k \hat{\pi}_k \cdot \hat{\Sigma}_k
\end{aligned}$$
where $\hat{\pi}_k$, $\hat{\mu}_k$ and $\hat{\Sigma}_k$ are the sample proportion, sample mean and sample variance for the kth class, respectively.

- Note that the binary class selection variable y_{nk} groups data from the same class.

3 – Application: Class prediction for new Data

- Let's apply the trained model: given a 'new' input $x_\bullet$, use Bayes rule to get posterior class probability
$$\begin{aligned}
p(\mathcal{C}_k|x_\bullet, \theta) &\propto p(\mathcal{C}_k)\, p(x_\bullet|\mathcal{C}_k) \\
&\propto \pi_k \exp\left\{ -\frac{1}{2} (x_\bullet - \mu_k)^T \Sigma^{-1} (x_\bullet - \mu_k) \right\} \\
&\propto \exp\left\{ \mu_k^T \Sigma^{-1} x_\bullet - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k \right\} \\
&= \exp\{\beta_k^T x + \gamma_k\}
\end{aligned}$$
where
$$\begin{aligned}
\beta_k &= \Sigma^{-1} \mu_k \\
\gamma_k &= -\frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k \,.
\end{aligned}$$

- The class posterior function
$$\phi(a_k) \triangleq \frac{\exp(a_k)}{\sum_{k'} \exp(a_{k'})}$$
is called a **softmax** function. Note that the softmax function is per definition properly normalized in the sense that $\sum_k \phi(a_k) = 1$.

Discrimination Boundaries

- The class log-posterior $\log p(\mathcal{C}_k|x) \propto \beta_k^T x + \gamma_k$ is a linear function of the input features.

- Thus, the contours of equal probability (**discriminant functions**) are lines (hyperplanes) in feature space"
$$\log \frac{p(\mathcal{C}_k|x, \theta)}{p(\mathcal{C}_j|x, \theta)} = \beta_{kj}^T x + \gamma_{kj} = 0$$
where we defined $\beta_{kj} \triangleq \beta_k - \beta_j$ and similarly for γ_{kj}.

- (homework). What happens if we had not assumed class-independent variances $\Sigma_k = \Sigma$? Are the discrimination functions still linear? quadratic?

- How to classify a new input $x_\bullet$? The Bayesian answer is a posterior distribution $p(\mathcal{C}_k|x_\bullet)$. If you must choose, then the class with maximum posterior class probability

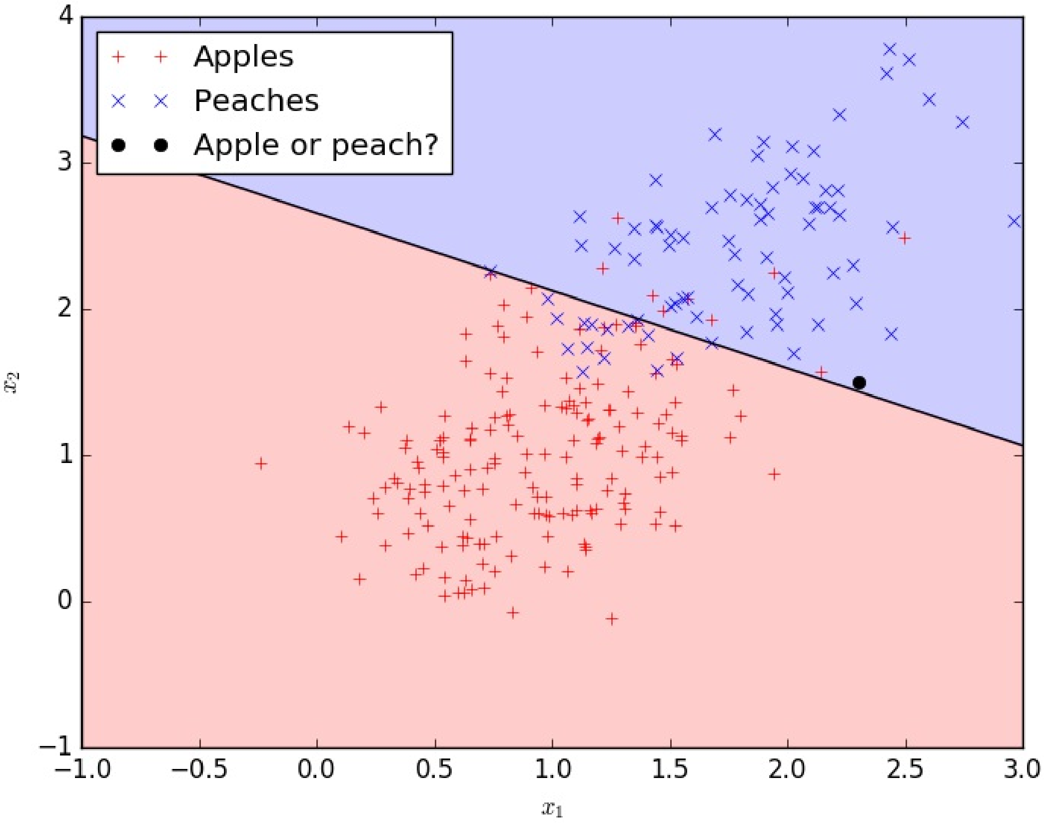
$$\begin{aligned} k^* &= \arg\max_k p(\mathcal{C}_k|x_\bullet) \\ &= \arg\max_k \left(\beta_k^T x_\bullet + \gamma_k\right) \end{aligned}$$

is an appealing decision.

CODE EXAMPLE

We'll apply the above results to solve the "apple or peach" example problem.

```
# Make sure you run the data-generating code cell first

# Multinomial (in this case binomial) density estimation
p_apple_est = sum(y.==true) / length(y)
π_hat = [p_apple_est; 1-p_apple_est]

# Estimate class-conditional multivariate Gaussian densities
d1 = fit_mle(FullNormal, X_apples')  # MLE density estimation d1 = N(μ₁, Σ₁)
d2 = fit_mle(FullNormal, X_peaches') # MLE density estimation d2 = N(μ₂, Σ₂)
Σ = π_hat[1]*cov(d1) + π_hat[2]*cov(d2) # Combine Σ₁ and Σ₂ into Σ
conditionals = [MvNormal(mean(d1), Σ); MvNormal(mean(d2), Σ)] # p(x|C)

# Calculate posterior class probability of x• (prediction)
function predict_class(k, X) # calculate p(Ck|X)
    norm = π_hat[1]*pdf(conditionals[1],X) + π_hat[2]*pdf(conditionals[2],X)
    return π_hat[k]*pdf(conditionals[k], X) ./ norm
end
println("p(apple|x=x•) = $(predict_class(1,x_test))")

# Discrimination boundary of the posterior (p(apple|x;D) = p(peach|x;D) = 0.5)
β(k) = inv(Σ)*mean(conditionals[k])
γ(k) = -0.5 * mean(conditionals[k])' * inv(Σ) * mean(conditionals[k]) + log(π_hat[k])
function discriminant_x2(x1)
    # Solve discriminant equation for x2
    β12 = β(1) - β(2)
    γ12 = (γ(1) - γ(2))[1,1]
    return -1*(β12[1]*x1 + γ12) ./ β12[2]
end
plot_fruit_dataset() # Plot dataset
x1 = linspace(-1,3,10)
plot(x1, discriminant_x2(x1), "k-") # Plot discrimination boundary
fill_between(x1, -1, discriminant_x2(x1), color="r", alpha=0.2)
fill_between(x1, discriminant_x2(x1), 4, color="b", alpha=0.2);
```




```
p(apple|x=x•) = 0.4346603348898282
```

Recap Generative Classification

- Model speccification: $p(x, \mathcal{C}_k|\,\theta) = \pi_k \cdot \mathcal{N}(x|\mu_k, \Sigma)$

- If the class-conditional distributions are Gaussian with equal covariance matrices across classes ($\Sigma_k = \Sigma$), then the discriminant functions are hyperplanes in feature space.

- ML estimation for $\{\pi_k, \mu_k, \Sigma\}$ breaks down to simple density estimation for Gaussian and multinomial.

- Posterior class probability is a softmax function
$$p(\mathcal{C}_k|x, \theta) \propto \exp\{\beta_k^T x + \gamma_k\}$$
where $\beta_k = \Sigma^{-1}\mu_k$ and $\gamma_k = -\frac{1}{2}\mu_k^T \Sigma^{-1}\mu_k + \log \pi_k$.

Discriminative Classification

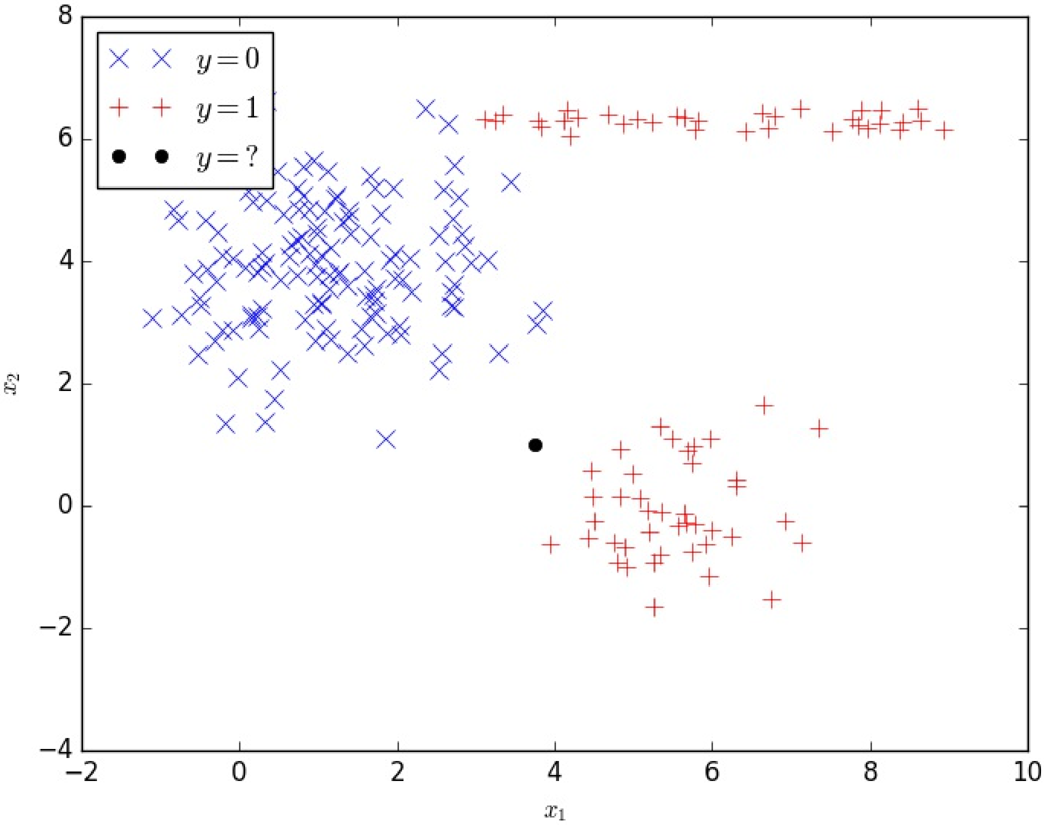
Preliminaries

- Goal
  - Introduction to discriminative classification models
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 203-206
    - T. Minka (2005), Discriminative models, not discriminative training

Problem: difficult class-conditional data distribitions

Our task will be the same as in the preceding class on (generative) classification. But this time, the class-conditional data distributions look very non-Gaussian, yet the linear discriminative boundary looks easy enough:

```
# Generate dataset {(x1,y1),...,(xN,yN)}
# x is a 2-d feature vector [x_1;x_2]
# y ∈ {false,true} is a binary class label
# p(x|y) is multi-modal (mixture of uniform and Gaussian distributions)
using PyPlot
include("scripts/lesson8_helpers.jl")
N = 200
X, y = genDataset(N) # Generate data set, collect in matrix X and vector y
X_c1 = X[:,find(.!y)]'; X_c2 = X[:,find(y)]' # Split X based on class label
X_test = [3.75; 1.0] # Features of 'new' data point
function plotDataSet()
    plot(X_c1[:,1], X_c1[:,2], "bx", markersize=8)
    plot(X_c2[:,1], X_c2[:,2], "r+", markersize=8, fillstyle="none")
    plot(X_test[1], X_test[2], "ko")
    xlabel(L"x_1"); ylabel(L"x_2"); legend([L"y=0", L"y=1",L"y=?"], loc=2)
    xlim([-2;10]); ylim([-4, 8])
end
plotDataSet();
```



Main Idea of Discriminative Classification

- Again, a data set is given by $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ with $x_n \in \mathbb{R}^D$ and $y_n \in \mathcal{C}_k$, with $k = 1, \ldots, K$.

- Sometimes, the precise assumptions of the (multinomial-Gaussian) generative model
$$p(x_n, \mathcal{C}_k|\theta) = \pi_k \cdot \mathcal{N}(x_n|\mu_k, \Sigma)$$
clearly do not match the data distribution.

- Here's an **IDEA**! Let's model the posterior
$$p(\mathcal{C}_k|x_n)$$
directly, without any assumptions on the class densities.

- Of course, this implies also that we build direct models for the **discrimination boundaries**
$$\log \frac{p(\mathcal{C}_k|x_n)}{p(\mathcal{C}_j|x_n)} \overset{!}{=} 0$$

1. Model Specification

- [Q.] What model should we use for $p(\mathcal{C}_k|x_n)$?

- [A.] Get inspiration from the generative approach: choose the familiar softmax structure **with linear discrimination bounderies** for the posterior class probability

$$p(\mathcal{C}_k|x_n,\theta) = \frac{e^{\theta_k^T x_n}}{\sum_j e^{\theta_j^T x_n}}$$

but **do not impose a Gaussian structure on the class features**.

- $\Rightarrow$ There are **two key differences** between the discriminative and generative approach:

  1. In the discriminative approach, the parameters θ_k are **not** structured into $\{\mu_k, \Sigma, \pi_k\}$. This provides discriminative approach with more flexibility.
  2. ML learning for the discriminative approach by optimization of *conditional* likelihood $\prod_n p(y_n|x_n,\theta)$ rather than *joint* likelihood $\prod_n p(y_n, x_n|\theta)$.

2. ML Estimation for Discriminative Classification

- The conditional log-likelihood for discriminative classification is

$$\mathrm{L}(\theta) = \log \prod_n \prod_k p(\mathcal{C}_k|x_n,\theta)^{y_{nk}}$$

- Computing the gradient $\nabla_{\theta_k} \mathrm{L}(\theta)$ (NB: revised text) leads to (for proof, see next slide)

$$\nabla_{\theta_k} \mathrm{L}(\theta) = \sum_n \Big(\underbrace{y_{nk}}_{\text{target}} - \underbrace{\frac{e^{\theta_k^T x_n}}{\sum_j e^{\theta_j^T x_n}}}_{\text{prediction}} \Big) \cdot x_n$$

- Compare this to the gradient for *linear* regression:

$$\nabla_\theta \mathrm{L}(\theta) = \sum_n \left(y_n - \theta^T x_n\right) x_n$$

- In both cases

$$\nabla_\theta \mathrm{L} = \sum_n \left(\text{target}_n - \text{prediction}_n\right) \cdot \text{input}_n$$

- The parameter vector θ for logistic regression can be estimated through iterative gradient-based adaptation. E.g. (with iteration index i),

$$\hat{\theta}^{(i+1)} = \hat{\theta}^{(i)} + \eta \cdot \nabla_\theta \mathrm{L}(\theta)\big|_{\theta=\hat{\theta}^{(i)}}$$

2. (OPTIONAL) Proof of Derivative of Log-likelihood for Discriminative Classification

- The Log-likelihood is $\mathrm{L}(\theta) = \log \prod_n \prod_k \underbrace{p(\mathcal{C}_k|x_n,\theta)}_{p_{nk}}{}^{y_{nk}} = \sum_{n,k} y_{nk} \log p_{nk}$

- Use the fact that the softmax $\phi_k \equiv e^{a_k} / \sum_j e^{a_j}$ has analytical derivative:

$$\begin{aligned}\frac{\partial \phi_k}{\partial a_j} &= \frac{(\sum_j e^{a_j})e^{a_k}\delta_{kj} - e^{a_j}e^{a_k}}{(\sum_j e^{a_j})^2} = \frac{e^{a_k}}{\sum_j e^{a_j}}\delta_{kj} - \frac{e^{a_j}}{\sum_j e^{a_j}}\frac{e^{a_k}}{\sum_j e^{a_j}} \\ &= \phi_k \cdot (\delta_{kj} - \phi_j)\end{aligned}$$

- Take the derivative of $\mathrm{L}(\theta)$ (or: how to spend a hour ...)

$$\begin{aligned}\nabla_{\theta_j} \mathrm{L}(\theta) &= \sum_{n,k} \frac{\partial \mathrm{L}_{nk}}{\partial p_{nk}} \cdot \frac{\partial p_{nk}}{\partial a_{nj}} \cdot \frac{\partial a_{nj}}{\partial \theta_j} \\ &= \sum_{n,k} \frac{y_{nk}}{p_{nk}} \cdot p_{nk}(\delta_{kj} - p_{nj}) \cdot x_n \\ &= \sum_n \Big(y_{nj}(1 - p_{nj}) - \sum_{k \neq j} y_{nk} p_{nj} \Big) \cdot x_n \\ &= \sum_n (y_{nj} - p_{nj}) \cdot x_n \\ &= \sum_n \Big(\underbrace{y_{nj}}_{\text{target}} - \underbrace{\frac{e^{\theta_j^T x_n}}{\sum_{j'} e^{\theta_{j'}^T x_n}}}_{\text{prediction}} \Big) \cdot x_n\end{aligned}$$

3. Application - Classify a new input

- Discriminative model-based prediction for a new input $x_\bullet$ is easy, namely substitute the ML estimate in the model to get

$$p(\mathcal{C}_k \,|\, x_\bullet, \hat{\theta}) = \frac{\exp\left(\hat{\theta}_k^T x_\bullet\right)}{\sum_{k'} \exp\left(\hat{\theta}_{k'}^T x_\bullet\right)} \propto \exp\left(\hat{\theta}_k^T x_\bullet\right)$$

- The contours of equal probability (**discriminant boundaries**) are lines (hyperplanes) in feature space given by

$$\log \frac{p(\mathcal{C}_k|x,\hat{\theta})}{p(\mathcal{C}_j|x,\hat{\theta})} = \left(\hat{\theta}_k - \hat{\theta}_j\right)^T x = 0$$

CODE EXAMPLE

Let us perform ML estimation of θ on the data set from the introduction. To allow an offset in the discrimination boundary, we add a constant 1 to the feature vector x. We only have to specify the (negative) log-likelihood and the gradient w.r.t. θ. Then, we use an off-the-shelf optimisation library to minimize the negative log-likelihood.

We plot the resulting maximum likelihood discrimination boundary. For comparison we also plot the ML discrimination boundary obtained from the generative Gaussian classifier from lesson 7.

```
using Optim # Optimization library

y_1 = zeros(length(y)) # class 1 indicator vector
y_1[y.==false] = 1.
X_ext = vcat(X, ones(1, length(y))) # Extend X with a row of ones to allow an offset in the discri
mination boundary

# Implement negative log-likelihood function
function negative_log_likelihood(θ::Vector)
    # Return negative log-likelihood: -L(θ)
    p_1 = 1.0 ./ (1.0 + exp.(-X_ext' * θ))   # P(C1|X,θ)
    return -sum(log.( (y_1.*p_1) + ((1.-y_1).*(1.-p_1))) ) # negative log-likelihood
end

# Use Optim.jl optimiser to minimize the negative log-likelihood function w.r.t. θ
results = optimize(negative_log_likelihood, zeros(3), LBFGS())
θ = results.minimizer

# Plot the data set and ML discrimination boundary
plotDataSet()
p_1(x) = 1.0 ./ (1.0 + exp(-([x;1.]' * θ)))
boundary(x1) = -1./θ[2] * (θ[1]*x1 + θ[3])
plot([-2.;10.], boundary([-2.;10.]), "k-");

# Also fit the generative Gaussian model from lesson 7 and plot the resulting discrimination bound
ary for comparison
generative_boundary = buildGenerativeDiscriminationBoundary(X, y)
plot([-2.;10.], generative_boundary([-2.;10.]), "k:");
legend([L"y=0";L"y=1";L"y=?";"Discr. boundary";"Gen. boundary"], loc=3);
```

Given $\hat{\theta}$, we can classify a new input $x_\bullet = [3.75, 1.0]^T$:

```
x_test = [3.75;1.0]
println("P(C1|x•,θ) = $(p_1(x_test))")
```

```
P(C1|x•,θ) = 0.6476513551215346
```

- The generative model gives a bad result because the feature distribution of one class is clearly non-Gaussian: the model does not fit the data well.

- The discriminative approach does not suffer from this problem because it makes no assumptions about the feature distribition $p(x|y)$, it just estimates the conditional class distribution $p(y|x)$ directly.

Recap Classification

	Generative	**Discriminative**
1	Like **density estimation**, model joint prob. $$p(\mathcal{C}_k)p(x\|\mathcal{C}_k) = \pi_k \mathcal{N}(\mu_k, \Sigma)$$	Like (linear) **regression**, model conditional $$p(\mathcal{C}_k\|x,\theta)$$
2	Leads to **softmax** posterior class probability $$p(\mathcal{C}_k\|x,\theta) = e^{\theta_k^T x}/Z$$ with **structured** θ	**Choose** also softmax posterior class probability $$p(\mathcal{C}_k\|x,\theta) = e^{\theta_k^T x}/Z$$ but now with 'free' θ
3	For Gaussian $p(x\|\mathcal{C}_k)$ and multinomial priors, $$\hat{\theta}_k = \begin{bmatrix} -\frac{1}{2}\mu_k^T \sigma^{-1}\mu_k + \log \pi_k \\ \sigma^{-1}\mu_k \end{bmatrix}$$ **in one shot**.	Find $\hat{\theta}_k$ through gradient-based adaptation $$\nabla_{\theta_k} \mathrm{L}(\theta) = \sum_n \left(y_{nk} - \frac{e^{\theta_k^T x_n}}{\sum_{k'} e^{\theta_{k'}^T x_n}} \right) x_n$$

Clustering with Gaussian Mixture Models

Preliminaries

- Goal
  - Introduction to the Expectation-Maximization (EM) Algorithm with application to Gaussian Mixture Models (GMM)
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 430-439 for Gaussian Mixture Models

Limitations of Simple IID Gaussian Models

Sofar, model inference was solved analytically, but we used strong assumptions:

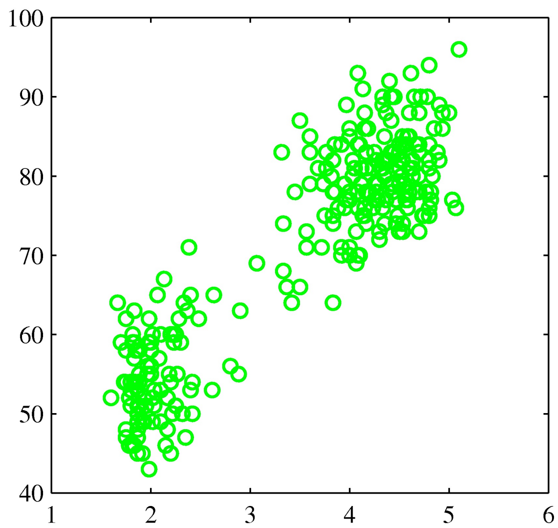
- IID sampling, $p(D) = \prod_n p(x_n)$
- Simple Gaussian (or multinomial) PDFs, $p(x_n) = \mathcal{N}(x_n|\mu, \Sigma)$
- Some limitations of Simple Gaussian Models with IID Sampling:
  - What if the PDF is **multi-modal** (or is just not Gaussian in any other way)?
  - Covariance matrix Σ has $D(D+1)/2$ parameters.
    - This quickly becomes **a very large number** for increasing dimension D.
  - Temporal signals are often **not IID**.

Towards More Flexible Models

- What if the PDF is multi-modal (or is just not Gaussian in any other way)?
  - **Discrete latent** variable models (a.k.a. **mixture** models). We'll cover this case in this lesson.
- Covariance matrix Σ has $D(D+1)/2$ parameters. This quickly becomes very large for increasing dimension D.
  - **Continuous latent** variable models (a.k.a. **dimensionality reduction** models). Covered in lesson 11.
- Temporal signals are often not IID.
  - Introduce **Markov dependencies** and **latent state** variable models. This will be covered in lesson 13.

Illustrative Example

- You're now asked to build a density model for a data set (Old Faithful, Bishop pg. 681) that clearly is not distributed as a single Gaussian:



Unobserved Classes

Consider again a set of observed data $D = \{x_1, \dots, x_N\}$

- This time we suspect that there are unobserved class labels that would help explain (or predict) the data, e.g.,
  - the observed data are the color of living things; the unobserved classes are animals and plants.
  - observed are wheel sizes; unobserved categories are trucks and personal cars.
  - observed is an audio signal; unobserved classes include speech, music, traffic noise, etc.

- Classification problems with unobserved classes are called **Clustering** problems. The learning algorithm needs to **discover the classes from the observed data**.

The Gaussian Mixture Model (GMM)

- If the categories were observed as well, these data could be nicely modeled by the previously discussed generative classification framework.

- Let us use **equivalent model assumptions to linear generative classification**:

$$p(x_n, \mathcal{C}_k) = \overbrace{p(\mathcal{C}_k)}^{\text{class prior}} \overbrace{p(x_n|\mathcal{C}_k)}^{\text{likelihood}} = \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)$$

where, as previously, we use notational shorthand $\mathcal{C}_k \triangleq (z_{nk} = 1)$ and a *hidden* 1-of-K selector variable z_{nk} with each observation x_n as

$$z_{nk} = \begin{cases} 1 & \text{if } z_n \in \mathcal{C}_k \\ 0 & \text{otherwise} \end{cases}$$

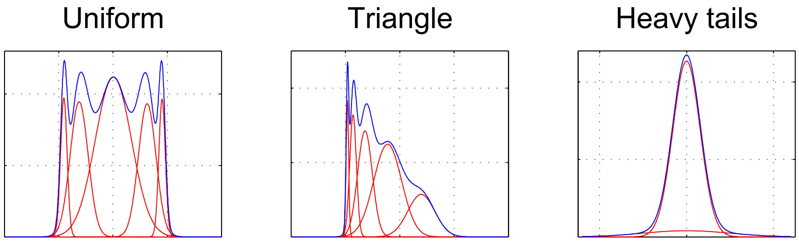
- Since the class labels are not observed, the probability for observations in a GMM is obtained through *marginalization* over the classes (this is different in classification problems, where the classes are observed):

$$p(x_n) = \sum_k \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)$$

  - The class priors $p(\mathcal{C}_k) = \pi_k$ are often called *mixture coefficients*.

Gaussian Mixture Models

- GMMs are very popular models. They have decent computational properties and are **universal approximators of densities** (as long as there are enough Gaussians of course)



Inference: Log-Likelihood for GMM

- The log-likelihood for observed data $D = \{x_1, \ldots, x_N\}$,

$$\begin{aligned}\log p(D|\theta) &\stackrel{\text{IID}}{=} \sum_n \log p(x_n|\theta) = \sum_n \log \left(\sum_{k=1}^{K} p(\mathcal{C}_k)\, p(x_n|\mathcal{C}_k) \right) \\ &= \sum_n \log \left(\sum_k \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k) \right)\end{aligned}$$

  ... and now the log-of-sum cannot be further simplified.

- Compare this to the log-likelihood for $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ in (generative) classification:

$$\begin{aligned}\log p(D|\theta) &= \sum_n \log \prod_k p(x_n, \mathcal{C}_k|\theta)^{y_{nk}} = \sum_{n,k} y_{nk} \log p(x_n, \mathcal{C}_k|\theta) \\ &= \sum_k m_k \log \pi_k + \sum_{n,k} y_{nk} \log \mathcal{N}(x_n|\mu_k, \Sigma)\end{aligned}$$

  which led to easy Gaussian and multinomial parameter estimation.

- Fortunately GMMs can be trained by maximum likelihood using an efficient algorithm: Expectation-Maximization.

Introducing a Soft Class Indicator

Let's introduce a 1-of-K selector variable z_{nk} to represent the *unobserved* classes $\mathcal{C}_k$ by

$$z_{nk} = \begin{cases} 1 & \text{if } x_n \text{ in class } \mathcal{C}_k \\ 0 & \text{otherwise} \end{cases}$$

- We don't *observe* the class labels z_{nk}, but we can compute the posterior probability for the class labels, given the observation x_n:

$$p(\mathcal{C}_k|x_n) = \frac{p(x_n|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_{k'} p(x_n|\mathcal{C}_{k'})p(\mathcal{C}_{k'})} = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{k'} \pi_{k'} \mathcal{N}(x_n|\mu_{k'}, \Sigma_{k'})}$$

- The $\gamma_{nk} \triangleq p(\mathcal{C}_k|x_n) = p(z_{nk} = 1|x_n)$ are also called **responsibilities**. Note that $0 \leq \gamma_{nk} \leq 1$ by definition and that they can be evaluated (i.e., we can compute it).

- The responsibilities γ_{nk} are soft class indicators and they play the same role in clustering as class selection variables (y_{nk}) do in classification problems.

ML estimation for Clustering: The Expectation-Maximization (EM) Algorithm Idea

- **IDEA**: Let's apply the (generative) classification formulas and substitute the reponsibilities γ_{nk} wherever the formulas use the binary class indicators y_{nk}.

- Try parameter updates (same as for generative Gaussian classification):

$$
\begin{aligned}
\hat{\pi}_k &= \frac{m_k}{N}\,, \quad \text{where } m_k = \sum_n \gamma_{nk} \\
\hat{\mu}_k &= \frac{1}{m_k} \sum_n \gamma_{nk} x_n \\
\hat{\Sigma}_k &= \frac{1}{m_k} \sum_n \gamma_{nk} (x_n - \hat{\mu}_k)(x_n - \hat{\mu}_k)^T
\end{aligned}
$$

- But wait ..., the responsibilities $\gamma_{nk} = \frac{\pi_k \mathcal{N}(x_n|\mu_k,\Sigma_k)}{\sum_j \pi_j \mathcal{N}(x_n|\mu_j,\Sigma_j)}$ are a function of the model parameters $\{\pi, \mu, \Sigma\}$ and the parameter updates depend on the responsibilities ...

- **Solution**: Iterate updating between γ_{nk} and the parameters $\{\pi, \mu, \Sigma\}$. This procedure is called the **Expectation-Maximization (EM)** algorithm.

CODE EXAMPLE

We'll perform clustering on the data set from the illustrative example by fitting a GMM consisting of two Gaussians using the EM algorithm.

```julia
using DataFrames, CSV
include("scripts/gmm_plot.jl") # Holds plotting function
old_faithful = CSV.read("datasets/old_faithful.csv")
X = convert(Matrix{Float64}, [old_faithful[1] old_faithful[2]]')
N = size(X, 2)

# Initialize the GMM. We assume 2 clusters.
clusters = [MvNormal([4.;60.], diagm([.5;10^2]));
            MvNormal([2.;80.], diagm([.5;10^2]))]
π_hat = [0.5; 0.5]                     # Mixing weights
γ = fill!(Matrix{Float64}(2,N), NaN)  # Responsibilities (row per cluster)

# Define functions for updating the parameters and responsibilities
function updateResponsibilities!(X, clusters, π_hat, γ)
    # Expectation step: update γ
    norm = [pdf(clusters[1], X) pdf(clusters[2], X)] * π_hat
    γ[1,:] = (π_hat[1] * pdf(clusters[1],X) ./ norm)'
    γ[2,:] = 1 - γ[1,:]
end
function updateParameters!(X, clusters, π_hat, γ)
    # Maximization step: update π_hat and clusters using ML estimation
    m = sum(γ, 2)
    π_hat = m / N
    μ_hat = (X * γ') ./ m'
    for k=1:2
        Z = (X .- μ_hat[:,k])
        Σ_k = Hermitian(((Z .* (γ[k,:])') * Z') / m[k])
        clusters[k] = MvNormal(μ_hat[:,k], convert(Matrix, Σ_k))
    end
end

# Execute the algorithm: iteratively update parameters and responsibilities
subplot(2,3,1); plotGMM(X, clusters, γ); title("Initial situation")
updateResponsibilities!(X, clusters, π_hat, γ)
subplot(2,3,2); plotGMM(X, clusters, γ); title("After first E-step")
updateParameters!(X, clusters, π_hat, γ)
subplot(2,3,3); plotGMM(X, clusters, γ); title("After first M-step")
iter_counter = 1
for i=1:3
    for j=1:i+1
        updateResponsibilities!(X, clusters, π_hat, γ)
        updateParameters!(X, clusters, π_hat, γ)
        iter_counter += 1
    end
    subplot(2,3,3+i); plotGMM(X, clusters, γ);
    title("After $(iter_counter) iterations")
end
PyPlot.tight_layout()
```

Note that you can step through the interactive demo yourself by running this script in julia. You can run a script in julia by `julia> include("path/to/script-name.jl")`

Clustering vs. (Generative) Classification

	Classification	Clustering
1	Class label y_n is observed	Class label z_n is latent
2	log-likelihood **conditions** on observed class $$\propto \sum_{nk} y_{nk} \log \mathcal{N}(x_n\|\mu_k,\Sigma_k)$$	log-likelihood **marginalizes** over latent classes $$\propto \sum_n \log \sum_k \pi_k \mathcal{N}(x_n\|\mu_k,\Sigma_k)$$
3	'Hard' class selector $$y_{nk} = \begin{cases} 1 & \text{if } y_n \text{ in class } \mathcal{C}_k \\ 0 & \text{otherwise} \end{cases}$$	'Soft' class responsibility $$\gamma_{nk} = p(\mathcal{C}_k\|x_n)$$
4	Estimation: $$\hat{\pi}_k = \frac{1}{N}\sum_n y_{nk}$$ $$\hat{\mu}_k = \frac{\sum_n y_{nk} x_n}{\sum_n y_{nk}}$$ $$\hat{\Sigma}_k = \frac{\sum_n y_{nk}(x_n-\hat{\mu}_k)(x_n-\hat{\mu}_k)^T}{\sum_n y_{nk}}$$	Estimation (1 update of M-step!) $$\hat{\pi}_k = \frac{1}{N}\sum_n \gamma_{nk}$$ $$\hat{\mu}_k = \frac{\sum_n \gamma_{nk} x_n}{\sum_n \gamma_{nk}}$$ $$\hat{\Sigma}_k = \frac{\sum_n \gamma_{nk}(x_n-\hat{\mu}_k)(x_n-\hat{\mu}_k)^T}{\sum_n \gamma_{nk}}$$

The General EM Algorithm

Preliminaries

- Goal
    - A more formal treatment of the general EM algorithm
- Materials
    - Mandatory
        - These lecture notes
    - Optional
        - Bishop pp. 55-57 for Jensen's inequality
        - Bishop pp. 439-443 for EM applied to GMM
        - Bishop pp. 450-455 for the general EM algorithm
        - Liang (2015) Technical Details about the EM algorithm
        - Bo and Batzoglou (2008) What is the expectation algorithm?

The Kullback-Leibler Divergence

- In order to prove that the EM algorithm works, we will need Gibbs inequality, which is a famous theorem in information theory.
- Definition: the **Kullback-Leibler divergence** (a.k.a. **relative entropy**) is a distance measure between two distributions q and p and is defined as

$$D_{\text{KL}}(q \parallel p) \triangleq \sum_z q(z) \log \frac{q(z)}{p(z)}$$

- Theorem: **Gibbs Inequality** (proof uses Jensen inquality):

$$D_{\text{KL}}(q \parallel p) \geq 0$$

  with equality only iff $p = q$.

- Note that the KL divergence is an asymmetric distance measure, i.e. in general

$$D_{\text{KL}}(q \parallel p) \neq D_{\text{KL}}(p \parallel q)$$

EM by maximizing a Lower Bound on the Log-Likelihood

- Consider a model for observations x, hidden variables z and tuning parameters θ. Note that, for **any** distribution $q(z)$, we can expand the log-likelihood as follows:

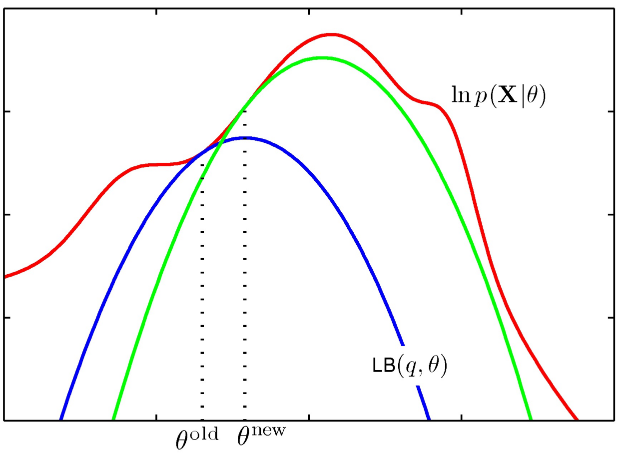
$$
\begin{aligned}
\mathrm{L}(\theta) &\triangleq \log p(x|\theta) \\
&= \sum_z q(z) \log p(x|\theta) \\
&= \sum_z q(z) \left(\log p(x|\theta) - \log \frac{q(z)}{p(z|x,\theta)} \right) + \sum_z q(z) \log \frac{q(z)}{p(z|x,\theta)} \\
&= \sum_z q(z) \log \frac{p(x,z|\theta)}{q(z)} + \underbrace{D_{\mathrm{KL}}\left(q(z) \,\|\, p(z|x,\theta)\right)}_{\text{Kullback-Leibler div.}} \\
&\geq \sum_z q(z) \log \frac{p(x,z|\theta)}{q(z)} \quad \text{(use Gibbs inequality)} \\
&= \underbrace{\sum_z q(z) \log p(x,z|\theta)}_{\text{expected complete-data log-likelihood}} + \underbrace{\mathcal{H}[q]}_{\text{entropy of } q} \\
&\triangleq \mathrm{LB}(q,\theta)
\end{aligned}
\tag{1}
$$

- Hence, $\mathrm{LB}(q,\theta)$ is a **lower bound** on the log-likelihood $\log p(x|\theta)$.

- Technically, the Expectation-Maximization (EM) algorithm is defined by coordinate ascent on $\mathrm{LB}(q,\theta)$:

$$
\begin{aligned}
&\text{Initialize}: \theta^{(0)} \\
&\text{for } m = 1,2,\ldots \text{until convergence} \\
&\quad q^{(m+1)} = \arg\max_q \mathrm{LB}(q,\theta^{(m)}) \qquad \% \text{ update responsibilities} \\
&\quad \theta^{(m+1)} = \arg\max_\theta \mathrm{LB}(q^{(m+1)},\theta) \qquad \% \text{ re-estimate parameters}
\end{aligned}
$$

- Since $\mathrm{LB}(q,\theta) \leq \mathrm{L}(\theta)$ (for all choices of $q(z)$), maximizing the lower-bound LB will also maximize the log-likelihood wrt θ. The *reason* to maximize LB rather than log-likelihood L directly is that $\arg\max \mathrm{LB}$ often leads to easier expressions. E.g., see this illustrative figure (Bishop p.453):



EM Details

Maximizing $\mathbf{LB}$ w.r.t. q

- Note that
$$\mathrm{LB}(q,\theta) = \mathrm{L}(\theta) - D_{\mathrm{KL}}\left(q(z) \parallel p(z|x,\theta)\right)$$
and consequenty, maximizing LB over q leads to minimization of the KL-divergence. Specifically, it follows from Gibbs inequality that
$$q^{(m+1)}(z) := p(z|x,\theta^{(m)})$$

Maximizing $\mathbf{LB}$ w.r.t. θ

- It also follows from (the last line of) the multi-line derivation above that maximizing LB w.r.t. θ amounts to maximization of the *expected complete-data log-likelihood* (where the complete data set is defined as $\{(x_i,z_i)\}_{i=1}^N$). Hence, the EM algorithm comprises iterations of
$$\mathbf{EM}: \; \theta^{(m+1)} := \underbrace{\arg\max_\theta}_{\text{M-step}} \underbrace{\sum_z \overbrace{p(z|x,\theta^{(m)})}^{=q^{(m+1)}(z)} \log p(x,z|\theta)}_{\text{E-step}}$$

Hoemwork exercise: EM for GMM Revisited

E-step

- Write down the GMM generative model
- The complete-data set is $D_c = \{x_1,z_1,x_2,z_2,\ldots,x_n,z_n\}$. Write down the *complete-data* likelihood $p(D_c|\theta)$
- Write down the complete-data *log*-likelihood $\log p(D_c|\theta)$
- Write down the *expected* complete-data log-likelihood $\mathrm{E}_Z\left[\log p(D_c|\theta)\right]$

M-step

- Maximize $\mathrm{E}_Z\left[\log p(D_c|\theta)\right]$ w.r.t. $\theta = \{\pi,\mu,\Sigma\}$
- Verify that your solution is the same as the 'intuitive' solution of the previous lesson.

Homework Exercise: EM for Three Coins problem

- You have three coins in your pocket. For each coin, outcomes $\in \{\mathrm{H},\mathrm{T}\}$.
$$p(\mathrm{H}) = \begin{cases} \lambda & \text{for coin 0} \\ \rho & \text{for coin 1} \\ \theta & \text{for coin 2} \end{cases}$$
- **Scenario**. Toss coin 0. If Head comes up, toss three times with coin 1; otherwise, toss three times with coin 2.
- The observed sequences **after** each toss with coin 0 were $\langle \mathrm{HHH}\rangle$, $\langle \mathrm{HTH}\rangle$, $\langle \mathrm{HHT}\rangle$, and $\langle \mathrm{HTT}\rangle$
- **Task**. Use EM to estimate most the likely values for λ, ρ and θ

Report Card on EM

- EM is a general procedure for learning in the presence of unobserved variables.

- In a sense, it is a **family of algorithms**. The update rules you will derive depend on the probability model assumed.

- (Good!) **No tuning parameters** such a learning rate, unlike gradient descent-type algorithms

- (Bad). EM is an iterative procedure that is very sensitive to initial conditions! EM converges to a **local optimum**.

- Start from trash $\rightarrow$ end up with trash. Hence, we need a good and fast initialization procedure (often used: K-Means, see Bishop p.424)

- Also used to train HMMs, etc.

(OPTIONAL SLIDE) The Free-Energy Principle

- The negative lower-bound $\mathrm{F}(q,\theta) \triangleq -\mathrm{LB}(q,\theta)$ also appears in various scientific disciplines. In statistical physics and variational calculus, F is known as the **free energy** functional. Hence, the EM algorithm is a special case of **free energy minimization**.

- It follows from Eq.1 above that

$$
\begin{aligned}
\mathrm{F}(q,\theta) &\triangleq \sum_z q(z) \log \frac{q(z)}{p(x,z|\theta)} \\
&= \underbrace{-\mathrm{L}(\theta)}_{-\text{log-likelihood}} + \underbrace{D_{\text{KL}}\left(q(z) \parallel p(z|x,\theta)\right)}_{\text{divergence}}
\end{aligned}
$$

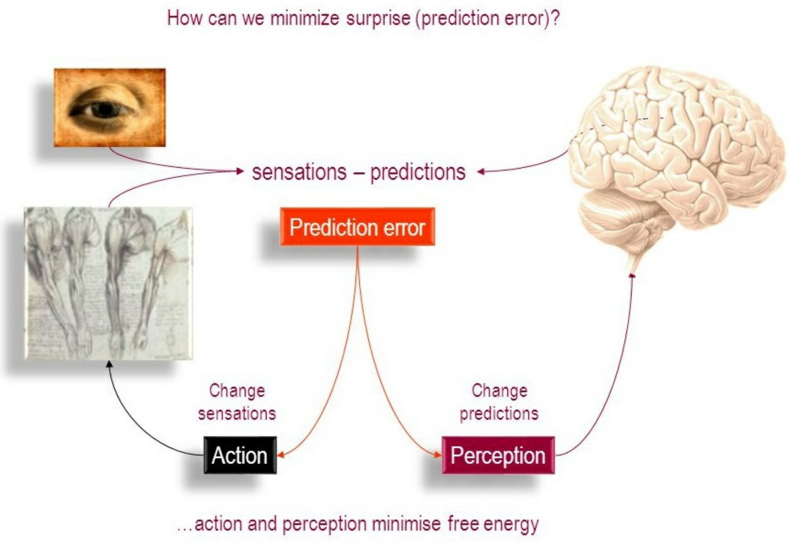
- The **Free-Energy Principle** (FEP) is an influential neuroscientific theory that claims that information processing in the brain is also an example of free-energy minimization, see Friston, 2009.

- According to FEP, the brain contains a generative model $p(x,z,a,\theta)$ for its environment. Here, x are the sensory signals (observations); z corresponds to (hidden) environmental causes of the sensorium; a represents our actions (control signals for movement) and θ describes the fixed 'physical rules' of the world.

(OPTIONAL SLIDE) The Free-Energy Principle, cont'd

- Solely through free-energy minimization, the brain infers an approximate posterior $q(z,a,\theta)$, thus inferring our **perception**, **actions**, and **learning** equations.

- Free-energy can be interpreted as (a generalized notion of sum of) prediction errors. Free-energy minimization aims to minimize prediction errors through perception, actions, and learning. (The next picture is from a 2012 tutorial presentation by Karl Friston)



(OPTIONAL SLIDE) The Free-Energy Principle, cont'd

- $\Rightarrow$ The brain is "nothing but" an approximate Bayesian agent that tries to build a model for its world. Actions (behavior) are selected to fulfill prior expectations about the world.

- In our BIASlab research team in the Signal Processing Systems group (FLUX floor 7), we work on developing (approximately Bayesian) **artificial** intelligent agents that learn purposeful behavior through interactions with their environments, inspired by the free-energy principle. Applications include
  - robotics
  - self-learning of new signal processing algorithms, e.g., for hearing aids
  - self-learning how to drive

- Let me know (email bert.de.vries@tue.nl) if you're interested in developing machine learning applications that are directly inspired by how the brain computes. We often have open traineeships or MSc thesis projects available.

Continuous Latent Variable Models – PCA and FA

Preliminaries

- Goal
  - Introduction to Linear Latent Variable Models on continuous domains, specifically **factor analysis** and **principal component analysis**
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 570-573, 577-580, 584-586 (PCA and FA)
    - M. Tipping and C. Bishop, Probabilistic Principal Component Analysis, Journal of the Royal Statistical Society. Series B, Vol.61, No.3, 1999

Continuous Latent Variable Models

- (Recall that) mixture models use a discrete class variable.

- Sometimes, it is more appropriate to think in terms of **continuous** underlying causes (factors) that control the observed data.
  - E.g., observe test results for subjects: English, Spanish and French

$$\underbrace{\begin{bmatrix} x_1\ (=\text{English}) \\ x_2\ (=\text{Spanish}) \\ x_3\ (=\text{French}) \end{bmatrix}}_{\text{observed}} = \begin{bmatrix} \lambda_{11}, \lambda_{12} \\ \lambda_{21}, \lambda_{22} \\ \lambda_{31}, \lambda_{32} \end{bmatrix} \cdot \underbrace{\begin{bmatrix} z_1\ (=\text{literacy}) \\ z_2\ (=\text{intelligence}) \end{bmatrix}}_{\text{causes}} + \underbrace{\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}}_{\text{noise}}$$

- (**Unsupervised Regression**). This is like (linear) regression with unobserved inputs.

Dimensionality Reduction

- If the dimension for the hidden 'causes' (z) is smaller than for the observed data (x), then the model (tries to) achieve **dimensionality reduction**.

- Key applications include
  1. **compression** (store z rather than x)
     - Compression through **real-valued** latent variables can be far more efficient than with discrete clusters.
     - E.g., with two 8-bit hidden factors, one can describe $2^{16} \approx 10^5$ settings; this would take 2^{16} clusters!
  2. **noise reduction** (e.g. in biomedical, financial or speech signals)
  3. **feature extraction** (e.g. as a pre-processor for classification)
  4. **visualization** (particularly if $\dim(Z) = 2$)

Example Problem: Visualization with missing data

- We consider 38 examples from the 18-dimensional data set from the **Tobamovirus** data set, see section 4.1 in Tipping and Bishop (1999) (Originally from Ripley (1996)).

- We will visualize this data set after projection onto the two principal axes (i.e., axes that explain largest data variance).

- We will also consider the visualization problem when **20% of the data set is missing**.

```
include("scripts/pca_demo_helpers.jl")
X = readDataSet("datasets/virus3.dat")
```

```
38×18 Array{Int64,2}:
 17  13  14  16   4  9  14  1  13  0  11  13  5   7  1  4  11  5
 12  11   9  12   6  5  12  1   9  1   7  12  5   6  0  4   8  2
 18  16  16  16   8  6  14  1  14  0   9  12  4   8  0  2  11  3
 18  16  15  19   8  6  11  1  15  1   7  13  5   8  0  2   9  3
 17  13  13  22   8  4  18  1  10  3   8  11  7   6  1  2  10  2
 16  13  16  21   9  3  17  1  10  4   7  12  7   5  1  2  11  3
 22  19  10  16  10  4  18  1  12  2   8  11  6   8  0  1   8  2
 20  10  24  10   6  9  21  0   7  0   7  18  4   9  1  4   8  2
 20  21  12  15   9  7  11  1   9  3   8  14  6   7  0  1  10  3
 20  21  12  15   9  7  11  1   9  3   9  14  5   7  0  1  10  3
 18  11  24  10   9  6  19  0  12  0   7  14  4  11  0  4   9  1
 20  12  23  10   8  5  20  0  13  0   6  13  4  11  0  4  10  1
 18  19  18  16   8  4  12  0  12  0  10  15  8   6  1  1  12  1
  ⋮                      ⋮                 ⋮                   ⋮
 17  12  22  10   8  5  18  0  14  0   5  13  4  10  0  3   9  1
 17  16  16  16   8  6  15  1  14  0   9  12  4   8  0  2  11  3
 19  17  15  17   7  6  15  1  14  0   8  12  4   8  0  2  10  3
 18  16  16  19   8  6  11  1  15  1   7  13  5   8  0  2   9  3
 18  17  15  17   8  6  15  1  14  0   8  12  4   8  0  3   9  3
 15  12  14  23   8  3  17  1   9  4   7  15  6   6  1  2  11  2
 13  11  14  22   7  3  17  1  10  4   8  13  6   6  1  3  11  2
 16  11  15  23  10  4  18  1  10  3   7  12  6   5  1  2   9  3
 14  11  14  25  11  3  19  2  10  2   7  12  6   5  1  2   9  3
 11  11  15  24  10  5  18  1  11  1   7  14  5   7  2  3  11  2
 15   9  12  21   8  4  21  1  10  3   7  15  7   6  1  3  10  3
 15  11  15  22   7  3  19  1   8  3   4  14  6   5  1  2  10  2
```

Model Specification for LC-LVM

- In this lesson, we focus on *Linear* Continuous Latent Variable Models (**LC-LVM**).

- Introduce observation vector $x \in \mathbb{R}^D$ and M-dimensional (with $M < D$) real-valued **latent factor** z:
$$\begin{aligned} x &= Wz + \mu + \epsilon \\ z &\sim \mathcal{N}(0, I) \\ \epsilon &\sim \mathcal{N}(0, \Psi) \end{aligned}$$
or equivalently
$$\begin{aligned} p(x|z) &= \mathcal{N}(x|\, Wz + \mu, \Psi) && \text{(likelihood)} \\ p(z) &= \mathcal{N}(z|\, 0, I) && \text{(prior)} \end{aligned}$$

- W is the so-called $(D \times M)$-dim **factor loading matrix**. The parameters of this model are given by $\theta = \{W, \mu, \Psi\}$.

- For interesting models, the **observation noise covariance matrix** Ψ is always **diagonal**.

- Note the similarity of the likelihood function in LC-LVM and linear regression:
$$p(y|x) = \mathcal{N}(y|\theta^T x, \sigma^2)$$

LC-LVM Analysis (1): The marginal distribution $p(x)$

- Since the product of Gaussians is Gaussian, both the joint $p(x,z) = p(x|z)p(z)$, the marginal $p(x)$ and the conditional $p(z|x)$ distributions are also Gaussian.

- The marginal distribution for the observed data is

$$p(x) = \mathcal{N}(x|\,\mu, WW^T + \Psi)$$

since the **mean** evaluates to

$$\begin{aligned} \mathrm{E}[x] &= \mathrm{E}[Wz + \mu + \epsilon] \\ &= W\mathrm{E}[z] + \mu + \mathrm{E}[\epsilon] \\ &= \mu \end{aligned}$$

and the **covariance** matrix is

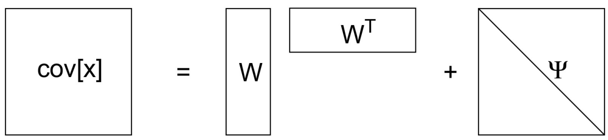
$$\begin{aligned} \mathrm{cov}[x] &= \mathrm{E}[(x-\mu)(x-\mu)^T] \\ &= \mathrm{E}[(Wz+\epsilon)(Wz+\epsilon)^T] \\ &= W\mathrm{E}[zz^T]W^T + \mathrm{E}[\epsilon\epsilon^T] \\ &= WW^T + \Psi \end{aligned}$$

- $\Rightarrow$ **LC-LVM is just a MultiVariate Gaussian (MVG) model** $x \sim \mathcal{N}(\mu, \Sigma)$ with the restriction that

$$\Sigma = WW^T + \Psi\,.$$

The Covariance Matrix of $p(x)$ is of Intermediate Complexity

- The effective covariance $\mathrm{cov}[x] = WW^T + \Psi$ is the low-rank outer product of two long skinny matrices plus a diagonal matrix.



- $\Rightarrow$ LC-LVM provides a MVG model of **intermediate complexity**. Compare the number of free parameters:
  - $D(D+1)/2$ for full Gaussian covariance Σ
  - $D(M+1)$ for LC-LVM model where $\Sigma = WW^T + \Psi$.
  - D for diagonal Gaussian covariance $\Sigma = \mathrm{diag}(\sigma_i^2)$
  - 1 for isotropic Gaussian noise $\Sigma = \sigma^2\mathrm{I}$

LC-LVM Analysis (2): The Factor Loading Matrix W is Not Unique

- The factor loading matrix W can only be estimated up to a rotation matrix R. Namely, if we rotate $W \rightarrow WR$, then the covariance matrix for observations x does not change (N.B.: a rotation (or orthogonal) matrix R is a matrix such that $R^T R = RR^T = I$):

$$WR(WR)^T + \Psi = WRR^T W^T + \Psi = WW^T + \Psi$$

- $\Rightarrow$ Two persons that estimate ML parameters for FA on the same data are **not guaranteed to find the same parameters**, since any rotation of W is equally likely.

- $\Rightarrow$ we can infer latent **subspaces** rather than individual components. One has to be careful when interpreting the numerical values of W and z.

LC-LVM analysis (3): Constraints on the Noise Variance Ψ

- When doing ML estimation for the parameters, a trivial solution for the covariance matrix $\Sigma_x = WW^T + \Psi$ is setting $\hat{W} = 0$ and $\hat{\Psi}$ equal to the sample variance of the data.

- In this case, all data correlation is explained as noise. (We'd like to avoid this.)

- $\Rightarrow$ The LC-LVM model is uninteresting without some restriction on the observation noise covariance matrix Ψ.

- The interesting cases are mostly for diagonal Ψ. Note that if Ψ is diagonal, all correlations between the (D) components of x **must be explained** by the rank-M matrix WW^T. Three model choices are common:

1. Factor Anaysis

- In Factor Analysis (**FA**), Ψ is restricted to be *diagonal*:

$$\Psi = \mathrm{diag}(\psi_i)$$

LC-LVM analysis (3): Constraints on the Noise Variance Ψ, cont'd

2. Probabilistic Principal Component Analysis

- In Probabilistic Principal Component Analysis (**pPCA**), the variances are further restricted to be the same,

$$\Psi = \sigma^2 I$$

3. Principal Component Analysis

- The 'regular' (deterministic) Principal Component Analysis (**PCA**) procedure can be obtained by further requiring that

$$\Psi = \lim_{\sigma^2 \to 0} \sigma^2 I$$
$$W^T W = I$$

  i.e., the noise model is discarded altogether and the columns of W are orthonormal.

- Regular PCA is a well-known deterministic procedure for dimensionality reduction (that predates pPCA).

$\Rightarrow$ FA, pPCA and PCA differ only by their model for the noise variance Ψ (namely, diagonal, isotropic and 'zeros').

Typical Applications

- In PCA (or pPCA), the noise variance is assumed to be the same for all components. This is appropriate if all components of the observed data are 'shifted' versions of each other.

$\Rightarrow$ **PCA is very widely applied to image and signal processing tasks!**

- Google (May-2015): [PCA "face recognition"] $>$ 300K hits; [PCA "noise reduction"] $>$ 100K hits

- FA is insensitive to scaling of individual components in the observed data (see appendix).
- Use FA if the data are not shifted versions of the same kind.

$\Rightarrow$ **FA has strong history in 'social sciences'**

ML estimation for pPCA Model

- Given the generative model for pPCA
$$\begin{aligned} p(x_n|z_n) &= \mathcal{N}(x_n \mid W z_n + \mu, \sigma^2 \mathrm{I}) \\ p(z_n) &= \mathcal{N}(z_n \mid 0, \mathrm{I}) \end{aligned}$$
and observations $D = \{x_1, \dots, x_N\}$, find ML estimates for the parameters $\theta = \{W, \mu, \sigma\}$

- **Inference for** μ is easy: x is a multivariate Gaussian with mean μ, so its ML estimate is
$$\hat{\mu} = \frac{1}{N} \sum_n x_n$$
Now subtract $\hat{\mu}$ from all data points ($x_n := x_n - \hat{\mu}$) and assume that we have zero-mean data.

- For ML estimation of W and σ^2, both gradient-ascent and EM are possible.

Solution method 1: Gradient-ascent on the log-likelihood

- Work out the gradients for the log-likelihood
$$\begin{aligned} &\log p(D|\theta) \\ &= -\frac{N}{2} \log |2\pi (W W^T + \sigma^2 \mathrm{I})| - \frac{1}{2} \sum_n x_n^T (W W^T + \sigma^2 \mathrm{I})^{-1} x_n \end{aligned}$$
and optimize w.r.t. W and σ^2.

- (Similarly to ML estimation in Gaussian mixture models), it turns out to be quite difficult to work out the gradient because of the coupling between W and σ^2 (but it is possible, see Tipping and Bishop, 1999).

Solution method 2: Use EM

- A big bonus for EM over gradient-based methods is that EM comfortably handles missing observations, e.g. through sensor malfunction. Missing observations are simply treated as hidden variables.

- Maximizing the *expected complete-data log-likelihood* leads to the following (see Bishop, pg.578 for derivation):

$$\begin{aligned} &\textbf{E-step}: \\ &M = W^T W + \sigma^2 \mathrm{I} \\ &\mathrm{E}[z_n] = M^{-1} W^T x_n \\ &\mathrm{E}\left[z_n z_n^T\right] = \sigma^2 M^{-1} + \mathrm{E}[z_n] \mathrm{E}[z_n]^T \end{aligned}$$

$$\begin{aligned} &\textbf{M-step}: \\ &W_{\text{new}} = \left[\sum_{n=1}^N x_n \mathrm{E}[z_n]^T\right] \left[\sum_{n=1}^N \mathrm{E}\left[z_n z_n^T\right]\right]^{-1} \\ &\sigma^2_{\text{new}} = \frac{1}{ND} \sum_{n=1}^N \left\{ x_n^T x_n - 2\mathrm{E}[z_n]^T W_{\text{new}}^T x_n + \mathrm{Tr}\left(\mathrm{E}\left[z_n z_n^T\right] W_{\text{new}}^T W_{\text{new}}\right) \right\} \end{aligned}$$

Solution method 2: Use EM, cont'd

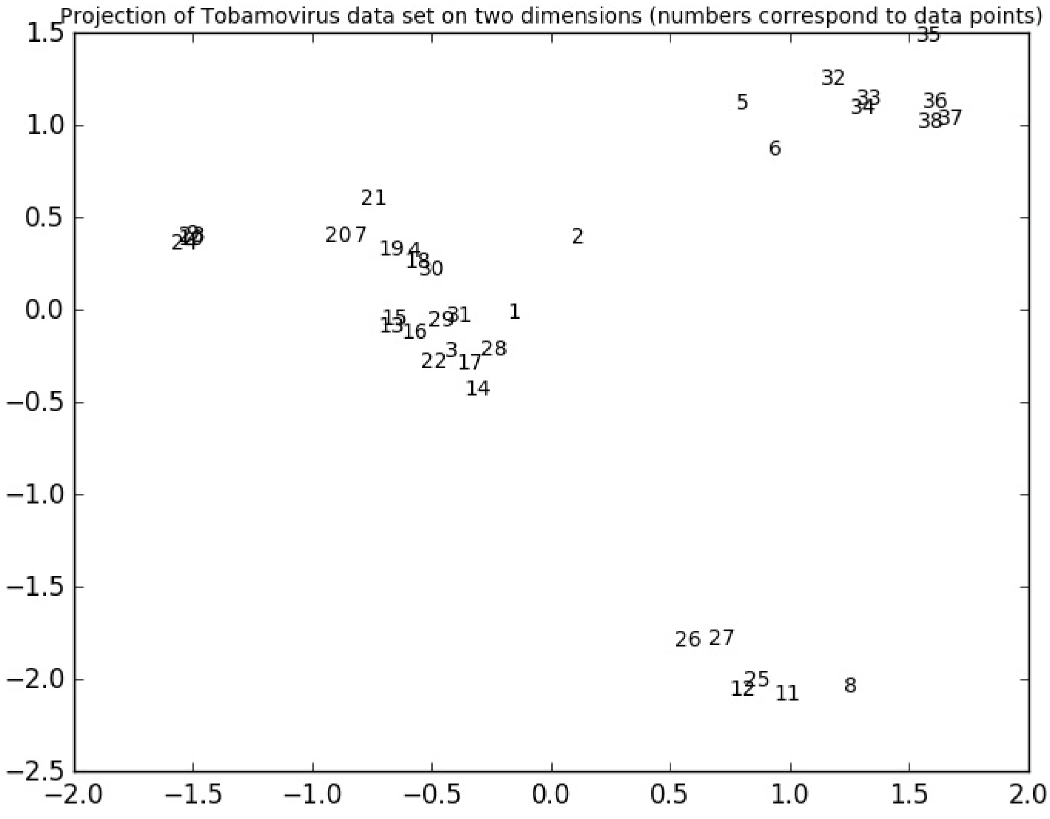
- Note that after x_n is observed, the unobserved 'input' z_n is not known exactly; the uncertainty about input z_n , as expressed by the covariance
$$\mathrm{cov}(z_n) = \mathrm{E}\left[z_n z_n^T\right] - \mathrm{E}\left[z_n\right]\mathrm{E}[z_n]^T = \sigma^2 M^{-1}$$
can be computed _before the data point x_n has been seen_.
  - Exercise: Show that the precision about z_n increases through observing x_n.
  - Compare this to linear regression, where we have full knowledge about an input-output pair.

- If there was no uncertainty about z_n , i.e., $\mathrm{E}\left[z_n\right] = z_n$ and $\mathrm{E}\left[z_n z_n^T\right] = z_n z_n^T$, then
$$W_{\text{new}} = \left[\sum_{n=1}^{N} x_n z_n^T\right]\left[\sum_{n=1}^{N} z_n z_n^T\right]^{-1}$$
  - Exercise: Verify that this solution resembles the [ML solution for linear regression] (http://nbviewer.ipython.org/github/bertdv/AIP-5SSB0/blob/master/lessons/notebooks/06_Linear-Regression.ipynb).

Example Problem Revisited

Let's perform pPCA on the example (**Tobamovirus**) data set using EM. We'll find the two principal components ($M = 2$), and then visualize the data in a 2-D plot. The implementation is quite straightforward, have a look at the source file if you're interested in the details.

```
(θ, Z) = pPCA(X', 2) # uses EM, implemented in scripts/pca_demo_helpers.jl. Feel free to try more/less dimensions.
using PyPlot
plot(Z[1,:]', Z[2,:]', "w")
for n=1:size(Z,2)
    PyPlot.text(Z[1,n], Z[2,n], string(n), fontsize=10) # put a label on the position of the data point
end
title("Projection of Tobamovirus data set on two dimensions (numbers correspond to data points)", fontsize=10);
```



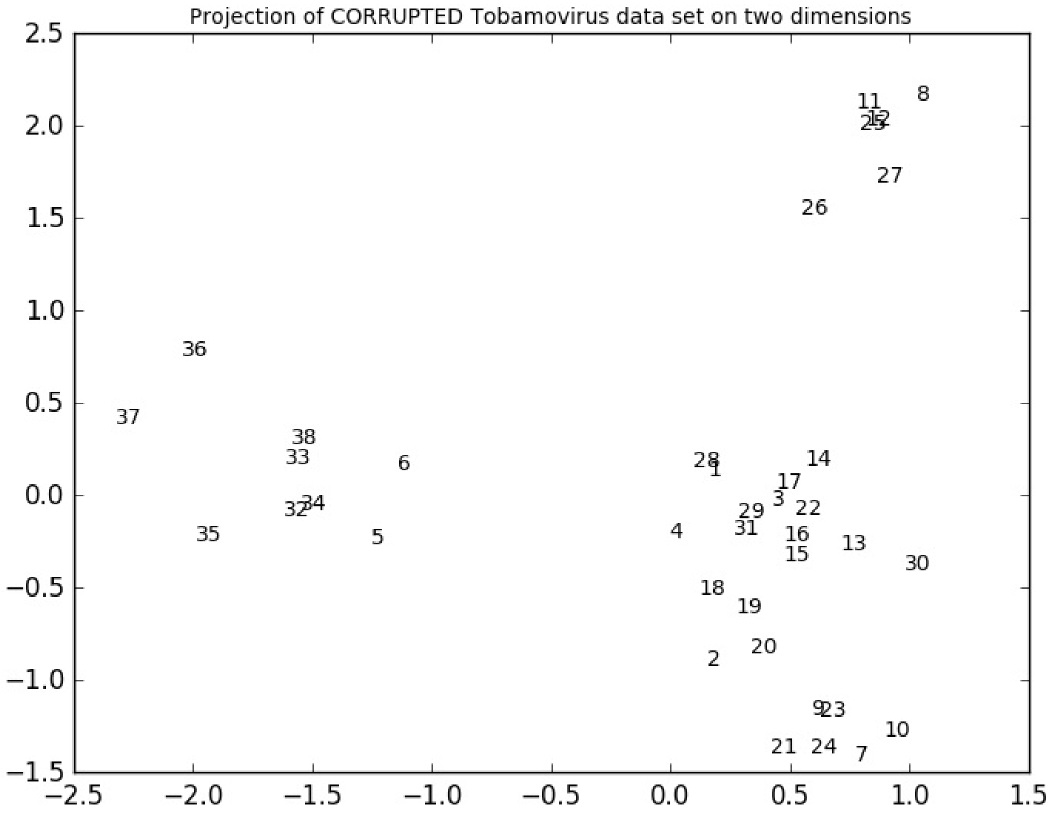
Note that the solution is not unique, but the clusters should be more or less persistent.

Now let's randomly remove 20% of the data:

```
X_corrupt = convert(Matrix{Float64}, X) # convert to floating point matrix so we can use NaN to in
dicate missing values
X_corrupt[rand(length(X)) .< 0.2] = NaN # set the elements of X_corrupt to NaN with probability 0.
2
X_corrupt
```

```
38×18 Array{Float64,2}:
  17.0   13.0  NaN     16.0  NaN    …    7.0  NaN    NaN     11.0    5.0
  12.0  NaN      9.0   12.0    6.0       6.0    0.0    4.0    8.0  NaN
  18.0   16.0   16.0   16.0    8.0       8.0  NaN      2.0  NaN      3.0
 NaN     16.0   15.0   19.0    8.0       8.0    0.0    2.0    9.0    3.0
 NaN     13.0   13.0   22.0  NaN         6.0    1.0  NaN     10.0    2.0
  16.0   13.0   16.0  NaN      9.0  …    5.0    1.0  NaN     11.0    3.0
  22.0   19.0   10.0   16.0   10.0       8.0    0.0    1.0    8.0    2.0
  20.0  NaN     24.0   10.0    6.0       9.0  NaN    NaN      8.0    2.0
  20.0  NaN     12.0   15.0    9.0     NaN    NaN      1.0  NaN      3.0
  20.0   21.0   12.0   15.0    9.0       7.0    0.0  NaN     10.0    3.0
  18.0   11.0   24.0   10.0    9.0  …   11.0    0.0    4.0    9.0    1.0
 NaN     12.0   23.0   10.0    8.0      11.0    0.0    4.0   10.0    1.0
  18.0   19.0   18.0  NaN    NaN         6.0    1.0    1.0   12.0    1.0
   ⋮                                ⋱                  ⋮
 NaN     12.0   22.0   10.0    8.0     NaN      0.0    3.0    9.0    1.0
  17.0  NaN    NaN     16.0  NaN         8.0    0.0    2.0   11.0    3.0
  19.0  NaN     15.0   17.0    7.0       8.0  NaN      2.0   10.0    3.0
  18.0  NaN     16.0  NaN    NaN         8.0  NaN    NaN    NaN      3.0
  18.0   17.0   15.0   17.0    8.0  …    8.0    0.0  NaN      9.0  NaN
  15.0   12.0  NaN     23.0    8.0       6.0    1.0  NaN     11.0    2.0
  13.0   11.0   14.0   22.0    7.0       6.0    1.0    3.0   11.0    2.0
  16.0   11.0  NaN     23.0  NaN         5.0    1.0    2.0    9.0    3.0
  14.0   11.0  NaN     25.0   11.0       5.0    1.0    2.0    9.0    3.0
  11.0  NaN     15.0   24.0   10.0  …    7.0  NaN      3.0   11.0    2.0
 NaN      9.0   12.0  NaN      8.0       6.0    1.0  NaN     10.0    3.0
  15.0   11.0   15.0   22.0    7.0       5.0    1.0    2.0   10.0    2.0
```

```
(θ, Z) = pPCA(X_corrupt', 2) # Perform pPCA on the corrupted data set
plot(Z[1,:]', Z[2,:]', "w")
for n=1:size(Z,2)
    PyPlot.text(Z[1,n], Z[2,n], string(n), fontsize=10) # put a label on the position of the data
point
end
title("Projection of CORRUPTED Tobamovirus data set on two dimensions", fontsize=10);
```



As you can see, pPCA is quite robust in the face of missing data.

Factor Graphs

Preliminaries

- Goal
  - Introduction to Forney-style factor graphs and message passing algorithms
- Materials
  - Mandatory
    - These lecture notes
    - Loeliger, 2007, pp. 1295-1300 (until section IV)
  - Optional
    - Video lecture by Frederico Wadehn (ETH Zurich) (**highly recommended**)

Why Factor Graphs?

- A probabilistic inference task gets its computational load mainly through the need for marginalization (i.e., computing integrals). E.g., for a generative model $p(x_1,x_2,x_3,x_4,x_5)$, the inference task $p(x_2|x_3)$ is given by

$$p(x_2|x_3) = \frac{\int p(x_1,x_2,x_3,x_4,x_5)\,\mathrm{d}x_1\,\mathrm{d}x_4\,\mathrm{d}x_5}{\int p(x_1,x_2,x_3,x_4,x_5)\,\mathrm{d}x_1\,\mathrm{d}x_2\mathrm{d}x_4\mathrm{d}x_5}$$

- Since these computations suffer from the "curse of dimensionality", we often need to solve a simpler problem in order to get an answer.

- Factor graphs provide an computationally efficient approach to solving inference problems **if the generative distribution can be factorized**.

- Factorization helps. For instance, if $p(x_1,x_2,x_3,x_4,x_5) = p(x_1)p(x_2,x_3)p(x_4)p(x_5|x_4)$, then

$$p(x_2|x_3) = \frac{\int p(x_1)p(x_2,x_3)p(x_4)p(x_5|x_4)\,\mathrm{d}x_1\mathrm{d}x_4\mathrm{d}x_5}{\int p(x_1)p(x_2,x_3)p(x_4)p(x_5|x_4)\,\mathrm{d}x_1\mathrm{d}x_2\mathrm{d}x_4\mathrm{d}x_5} = \frac{p(x_2,x_3)}{\int p(x_2,x_3)\mathrm{d}x_2}$$

  which is computationally much cheaper than the general case above.

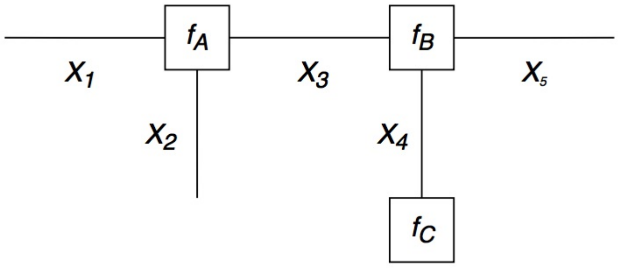
- In this lesson, we discuss how computationally efficient inference in factorized probability distributions can be automated.

Factor Graph Construction Rules

- Consider a function

$$f(x_1,x_2,x_3,x_4,x_5) = f_a(x_1,x_2,x_3)\cdot f_b(x_3,x_4,x_5)\cdot f_c(x_4)$$

- The factorization of this function can be graphically represented by a **Forney-style Factor Graph** (FFG):



- An FFG is an **undirected** graph subject to the following construction rules (Forney, 2001)

  1. A **node** for every factor;
  2. An **edge** (or **half-edge**) for every variable;
  3. Node g is connected to edge x **iff** variable x appears in factor g.

Some FFG Terminology

- f is called the **global function** and $f_\bullet$ are the **factors**.
- A **configuration** is an assigment of values to all variables.
- The **configuration space** is the set of all configurations, i.e., the domain of f
- A configuration $\omega = (x_1, x_2, x_3, x_4, x_5)$ is said to be **valid** iff $f(\omega) \neq 0$

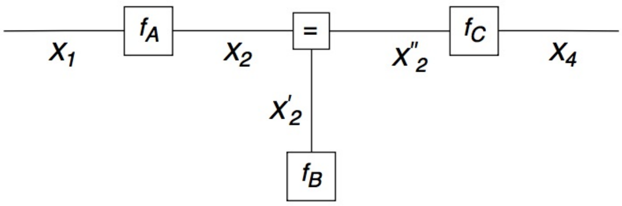
Equality Nodes for Branching Points

- Note that a variable can appear in maximally two factors in an FFG (since an edge has only two end points).

- Consider the factorization (where x_2 appears in three factors)

$$f(x_1, x_2, x_3, x_4) = f_a(x_1, x_2) \cdot f_b(x_2, x_3) \cdot f_c(x_2, x_4)$$

- For the factor graph representation, we will instead consider the function g, defined as
$$g(x_1, x_2, x_2', x_2'', x_3, x_4) = f_a(x_1, x_2) \cdot f_b(x_2', x_3) \cdot f_c(x_2'', x_4) \cdot f_=(x_2, x_2', x_2'')$$
where
$$f_=(x_2, x_2', x_2'') \triangleq \delta(x_2 - x_2')\, \delta(x_2 - x_2'')$$



Equality Nodes for Branching Points, cont'd

- Note that through introduction of auxiliary variables X_2' and X_2'' each variable in g appears in maximally two factors.

- The constraint $f_=(x, x', x'')$ enforces that $X = X' = X''$ **for every valid configuration**.

- Since f is a marginal of g, i.e.,

$$f(x_1,x_2,x_3,x_4) = \int g(x_1,x_2,x_2',x_2'',x_3,x_4)\,\mathrm{d}x_2'\mathrm{d}x_2''$$

it follows that any inference problem on f can be executed by a corresponding inference problem on g, e.g.,

$$\begin{aligned} f(x_1 \mid x_2) &\triangleq \frac{\int f(x_1,x_2,x_3,x_4)\,\mathrm{d}x_3\mathrm{d}x_4}{\int f(x_1,x_2,x_3,x_4)\,\mathrm{d}x_1\mathrm{d}x_3\mathrm{d}x_4} \\ &= \frac{\int g(x_1,x_2,x_2',x_2'',x_3,x_4)\,\mathrm{d}x_2'\mathrm{d}x_2''\mathrm{d}x_3\mathrm{d}x_4}{\int g(x_1,x_2,x_2',x_2'',x_3,x_4)\,\mathrm{d}x_1\mathrm{d}x_2'\mathrm{d}x_2''\mathrm{d}x_3\mathrm{d}x_4} \\ &\triangleq g(x_1 \mid x_2) \end{aligned}$$

- $\Rightarrow$ **Any factorization of a global function f can be represented by a Forney-style Factor Graph**.

Probabilistic Models as Factor Graphs

- FFGs can be used to express conditional independence (factorization) in probabilistic models.

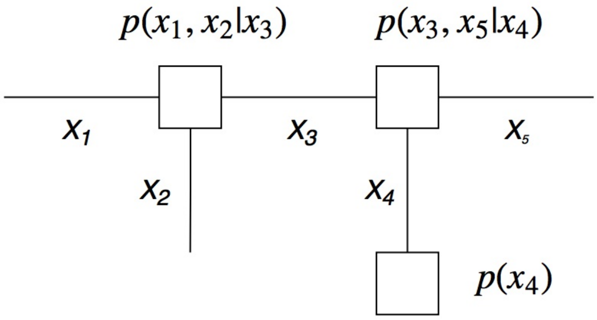
- For example, the (previously shown) graph for $f_a(x_1,x_2,x_3)\cdot f_b(x_3,x_4,x_5)\cdot f_c(x_4)$ could represent the probabilistic model

$$p(x_1,x_2,x_3,x_4,x_5) = p(x_1,x_2|x_3)\cdot p(x_3,x_5|x_4)\cdot p(x_4)$$

where we identify

$$\begin{aligned} f_a(x_1,x_2,x_3) &= p(x_1,x_2|x_3) \\ f_b(x_3,x_4,x_5) &= p(x_3,x_5|x_4) \\ f_c(x_4) &= p(x_4) \end{aligned}$$

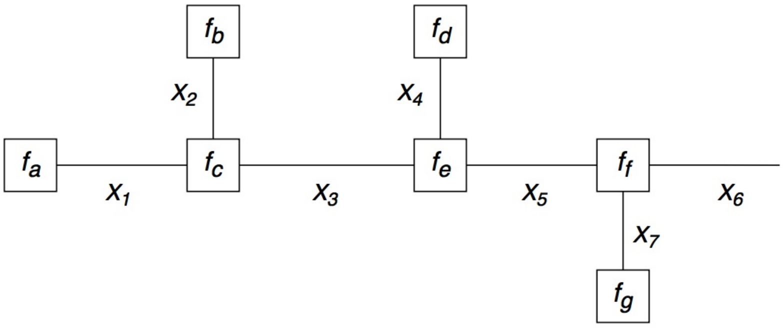
- This is the graph



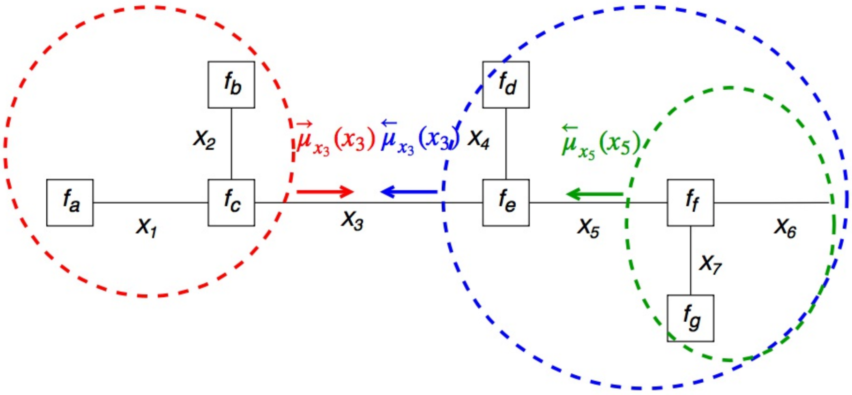
Inference by Closing Boxes

- Factorizations provide opportunities to cut on the amount of needed computations when doing inference. In what follows, we will use FFGs to process these opportunities in an automatic way (i.e., by message passing).

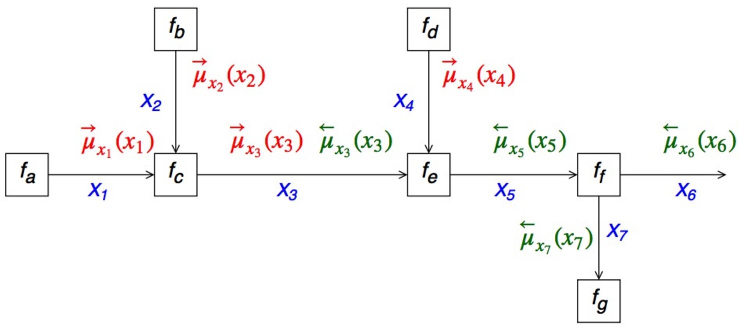
- Assume we wish to compute the marginal
$$\bar f(x_3) = \sum_{x_1, x_2, x_4, x_5, x_6, x_7} f(x_1, x_2, \ldots, x_7)$$
where f is factorized as given by the following FFG



- Due to the factorization, we can decompose this sum by the **distributive law** as
$$\bar f(x_3) = \underbrace{\left(\sum_{x_1,x_2} f_a(x_1)\, f_b(x_2)\, f_c(x_1,x_2,x_3)\right)}_{\overrightarrow{\mu}_{X_3}(x_3)} \cdot \underbrace{\left(\sum_{x_4,x_5} f_d(x_4)\, f_e(x_3,x_4,x_5) \cdot \underbrace{\left(\sum_{x_6,x_7} f_f(x_5,x_6,x_7)\, f_g(x_7)\right)}_{\overleftarrow{\mu}_{X_5}(x_5)}\right)}_{\overleftarrow{\mu}_{X_3}(x_3)}$$
which is computationally (much) lighter than executing the full sum $\sum_{x_1,\ldots,x_7} f(x_1, x_2, \ldots, x_7)$



- Messages may flow in both directions on any edge (here on X_3). We often draw *directed edges* in a FFG in order to distinguish forward messages $\overrightarrow{\mu}_\bullet(\cdot)$ (in the same direction as the arrow of the edge) from backward messages $\overleftarrow{\mu}_\bullet(\cdot)$ (in opposite direction). With directed edges, the FFG looks as follows:



- Crucially, drawing arrows on edges is only meant as a notational convenience. Technically, an FFG is an undirected graph.

- Note that $\overleftarrow{\mu}_{X_5}(x_5)$ is obtained by multiplying all enclosed factors (f_f, f_g) by the green dashed box, followed by marginalization over all enclosed variables (x_6, x_7).

- This is the **Closing the Box**-rule, which is a general recipe for marginalization of hidden variables and leads to a new factor with outgoing (sum-product) message

$$\mu_{\text{SP}} = \sum_{\substack{\text{enclosed} \\ \text{variables}}} \prod_{\substack{\text{enclosed} \\ \text{factors}}}$$

Evaluating the closing-the-box rule for individual nodes

- First, closing a box around the **terminal nodes** leads to $\overrightarrow{\mu}_{X_1}(x_1) \triangleq f_a(x_1)$, $\overrightarrow{\mu}_{X_2}(x_2) \triangleq f_b(x_2)$ etc.
  - So, the message out of a terminal node is the factor itself.
  - (Exercise) Derive now that the message coming from the open end of a half-edge always equals 1.

- The messages from **internal nodes** evaluate to:

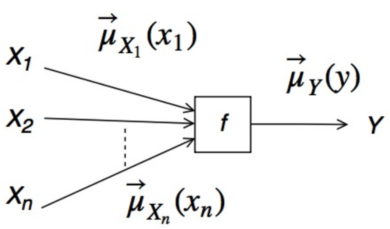
$$\begin{aligned}
\overrightarrow{\mu}_{X_3}(x_3) &= \sum_{x_1,x_2} f_a(x_1)\, f_b(x_2)\, f_c(x_1,x_2,x_3) \\
&= \sum_{x_1,x_2} \overrightarrow{\mu}_{X_1}(x_1) \overrightarrow{\mu}_{X_2}(x_2)\, f_c(x_1,x_2,x_3) \\
\overleftarrow{\mu}_{X_5}(x_5) &= \sum_{x_6,x_7} f_f(x_5,x_6,x_7)\, f_g(x_7) \\
&= \sum_{x_6,x_7} \overrightarrow{\mu}_{X_7}(x_7)\, f_f(x_5,x_6,x_7)
\end{aligned}$$

- Crucially, all message update rules can be computed from information that is **locally available** at each node.

Sum-Product Algorithm

- (**Sum-Product update rule**). This recursive pattern for computing messages applies generally and is called the **Sum-Product update rule**, which is really just a special case of the closing-the-box rule: For any node, the outgoing message is obtained by taking the product of all incoming messages and the node function, followed by summing out (marginalization) all incoming variables. What is left (the outgoing message) is a function of the outgoing variable only:

$$\overrightarrow{\mu}_Y(y) = \sum_{x_1,\ldots,x_n} \overrightarrow{\mu}_{X_1}(x_1) \cdots \overrightarrow{\mu}_{X_n}(x_n)\, f(y,x_1,\ldots,x_n)$$



- (**Sum-Product Theorem**). If the factor graph for a function f has **no cycles**, then the marginal $\bar{f}(x_3) = \sum_{x_1,x_2,x_4,x_5,x_6,x_7} f(x_1,x_2,\ldots,x_7)$ is given by the Sum-Product Theorem:

$$\bar{f}(x_3) = \overrightarrow{\mu}_{X_3}(x_3) \cdot \overleftarrow{\mu}_{X_3}(x_3)$$

- (**Sum-Product Algorithm**). If folows that the marginal $\bar{f}(x_3) = \sum_{x_1,x_2,x_4,x_5,x_6,x_7} f(x_1,x_2,\ldots,x_7)$ can be efficiently computed through sum-product messages. Executing inference through SP message passing is called the **Sum-Product Algorithm**.

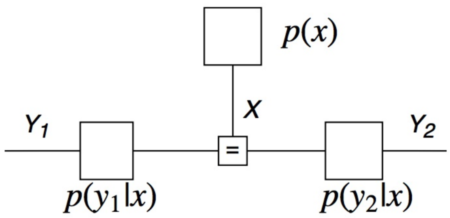
- (Exercise) Verfiy for yourself that all maginals in a cycle-free graph (a tree) can be computed exactly by starting with messages at the terminals and working towards the root of the tree.

Processing Observations in a Factor Graph

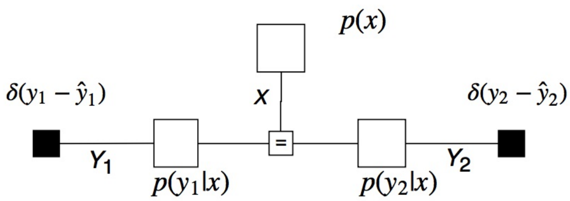
- Terminal nodes can be used describe **observed variables**, e.g., use a factor
$$f_Y(y) = \delta(y-3)$$
to terminate the edge for variable Y if $y=3$ is observed.

Example

- Consider a generative model
$$p(x,y_1,y_2) = p(x)\,p(y_1|x)\,p(y_2|x)\,.$$
  - This model expresses the assumption that Y_1 and Y_2 are independent measurements of X.



- Assume that we are interested in the posterior for X after observing $Y_1 = \hat{y}_1$ and $Y_2 = \hat{y}_2$. The posterior for X can be inferred by applying the sum-product algorithm to the following graph:



- (Note that) we usually draw terminal nodes for observed variables in the graph by smaller solid-black squares. This is just to help the visualization of the graph, since the computational rules are no different than for other nodes.

- (Exercise) Can you draw the messages that infer $p(x \mid y_1, y_2)$?

CODE EXAMPLE

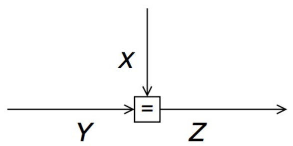
We'll use ForneyLab, a factor graph toolbox for Julia, to build the above graph, and perform sum-product message passing to infer the posterior $p(x|y_1,y_2)$. We assume $p(y_1|x)$ and $p(y_2|x)$ to be Gaussian likelihoods with known variances:
$$p(y_1 \mid x) = \mathcal{N}(y_1 \mid x, v_{y1})$$
$$p(y_2 \mid x) = \mathcal{N}(y_2 \mid x, v_{y2})$$
Under this model, the posterior is given by:
$$\begin{aligned} p(x \mid y_1, y_2) &\propto \overbrace{p(y_1 \mid x)\,p(y_2 \mid x)}^{\text{likelihood}}\overbrace{p(x)}^{\text{prior}} \\ &= \mathcal{N}(x \mid \hat{y}_1, v_{y1})\,\mathcal{N}(x \mid \hat{y}_2, v_{y2})\,\mathcal{N}(x \mid m_x, v_x) \end{aligned}$$
so we can validate the answer by solving the Gaussian multiplication manually.

```julia
using ForneyLab # version 0.7.1

# Data
y1_hat = 1.0
y2_hat = 2.0

# Construct the factor graph
fg = FactorGraph()
@RV x  ~ GaussianMeanVariance(constant(0.0), constant(4.0), id=:x) # Node p(x)
@RV y1 ~ GaussianMeanVariance(x, constant(1.0))            # Node p(y1|x)
@RV y2 ~ GaussianMeanVariance(x, constant(2.0))            # Node p(y2|x)
Clamp(y1, y1_hat)                   # Terminal (clamp) node for y1
Clamp(y2, y2_hat)                   # Terminal (clamp) node for y2
# draw(fg) # draw the constructed factor graph

# Perform sum-product message passing
eval(parse(sumProductAlgorithm(x, name="_algo1"))) # Automatically derives a message passing schedule
x_marginal = step_algo1!(Dict())[:x] # Execute algorithm and collect marginal distribution of x
println("Sum-product message passing result: p(x|y1,y2) = $(x_marginal)")

# Calculate mean and variance of p(x|y1,y2) manually by multiplying 3 Gaussians (see lesson 4 for details)
v = 1 / (1/4 + 1/1 + 1/2)
m = v * (0/4 + y1_hat/1.0 + y2_hat/2.0)
println("Manual result: p(x|y1,y2) = N(m=$(round(m,2)), V=$(round(v,2)))")
```

```
Sum-product message passing result: p(x|y1,y2) = (m=1.14, v=0.57)

Manual result: p(x|y1,y2) = N(m=1.14, V=0.57)
```

Example: SP Messages for the Equality Node

- Let´s compute the SP messages for the **equality node** $f_=(x,y,z) = \delta(z-x)\delta(z-y)$:



$$\begin{aligned}
\overrightarrow{\mu}_Z(z) &= \int \overrightarrow{\mu}_X(x)\,\overrightarrow{\mu}_Y(y)\,\delta(z-x)\delta(z-y)\,\mathrm{d}x\mathrm{d}y \\
&= \overrightarrow{\mu}_X(z) \int \overrightarrow{\mu}_Y(y)\,\delta(z-y)\,\mathrm{d}y \\
&= \overrightarrow{\mu}_X(z)\,\overrightarrow{\mu}_Y(z)
\end{aligned}$$

- By symmetry, this also implies (for the same equality node) that

$$\begin{aligned}
\overleftarrow{\mu}_X(x) &= \overrightarrow{\mu}_Y(x)\overleftarrow{\mu}_Z(x) \quad \text{and} \\
\overleftarrow{\mu}_Y(y) &= \overrightarrow{\mu}_X(y)\overleftarrow{\mu}_Z(y)\,.
\end{aligned}$$

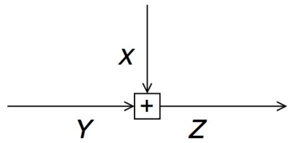
- Let us now consider the case of Gaussian messages $\overrightarrow{\mu}_X(x) = \mathcal{N}(\overrightarrow{m}_X, \overrightarrow{V}_X)$, $\overrightarrow{\mu}_Y(y) = \mathcal{N}(\overrightarrow{m}_Y, \overrightarrow{V}_Y)$ and $\overrightarrow{\mu}_Z(z) = \mathcal{N}(\overrightarrow{m}_Z, \overrightarrow{V}_Z)$. Let´s also define the precision matrices $\overrightarrow{W}_X \triangleq \overrightarrow{V}_X^{-1}$ and similarly for Y and Z. Then applying the SP update rule leads to multiplication of two Gaussian distributions, resulting in

$$\begin{aligned}
\overrightarrow{W}_Z &= \overrightarrow{W}_X + \overrightarrow{W}_Y \\
\overrightarrow{W}_Z \overrightarrow{m}_z &= \overrightarrow{W}_X \overrightarrow{m}_X + \overrightarrow{W}_Y \overrightarrow{m}_Y
\end{aligned}$$

- It follows that **message passing through an equality node is similar to applying Bayes rule**, i.e., fusion of two information sources. Does this make sense?

(OPTIONAL SLIDE) Example: SP Messages for the Addition Nodes

- Next, consider an **addition node** $f_+(x,y,z) = \delta(z-x-y)$:



$$\begin{aligned}\overrightarrow{\mu}_Z(z) &= \int \overrightarrow{\mu}_X(x)\,\overrightarrow{\mu}_Y(y)\,\delta(z-x-y)\,\mathrm{d}x\mathrm{d}y \\ &= \int \overrightarrow{\mu}_X(z)\,\overrightarrow{\mu}_Y(z-x)\,\mathrm{d}x\,,\end{aligned}$$

i.e., $\overrightarrow{\mu}_Z$ is the convolution of the messages $\overrightarrow{\mu}_X$ and $\overrightarrow{\mu}_Y$.

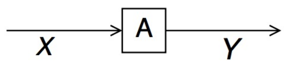
- Of course, for Gaussian messages, these update rules evaluate to

$$\overrightarrow{m}_Z = \overrightarrow{m}_X + \overrightarrow{m}_Y\ , \text{ and } \overrightarrow{V}_z = \overrightarrow{V}_X + \overrightarrow{V}_Y\,.$$

- **Exercise**: For the same summation node, work out the SP update rule for the *backward* message $\overleftarrow{\mu}_X(x)$ as a function of $\overrightarrow{\mu}_Y(y)$ and $\overleftarrow{\mu}_Z(z)$? And further refine the answer for Gaussian messages.

(OPTIONAL SLIDE) Example: SP Messages for Multiplication Nodes

- Next, let us consider a **multiplication** by a fixed (invertable matrix) gain $f_A(x,y) = \delta(y-Ax)$



$$\overrightarrow{\mu}_Y(y) = \int \overrightarrow{\mu}_X(x)\,\delta(y-Ax)\,\mathrm{d}x = \overrightarrow{\mu}_X(A^{-1}y)\,.$$

- For a Gaussian message input message $\overrightarrow{\mu}_X(x) = \mathcal{N}(\overrightarrow{m}_X, \overrightarrow{V}_X)$, the output message is also Gaussian with

$$\overrightarrow{m}_Y = A\overrightarrow{m}_X\ , \text{ and } \overrightarrow{V}_Y = A\overrightarrow{V}_X A^T$$

since

$$\begin{aligned}\acute{\mu}_Y(y) &= \overrightarrow{\mu}_X(A^{-1}y) \\ &\propto \exp\left(-\frac{1}{2}\left(A^{-1}y - \overrightarrow{m}_X\right)^T \overrightarrow{V}_X^{-1}\left(A^{-1}y - \overrightarrow{m}_X\right)\right) \\ &= \exp\left(-\frac{1}{2}\left(y - A\overrightarrow{m}_X\right)^T A^{-T}\overrightarrow{V}_X^{-1} A\left(y - A\overrightarrow{m}_X\right)\right) \\ &\propto \mathcal{N}(A\overrightarrow{m}_X, A\overrightarrow{V}_X A^T)\,.\end{aligned}$$

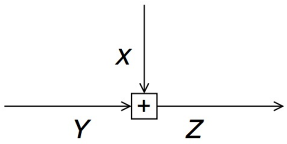
- **Excercise**: Proof that, for the same factor $\delta(y-Ax)$ and Gaussian messages, the (backward) sum-product message $\overleftarrow{\mu}_X$ is given by

$$\begin{aligned}\overleftarrow{\xi}_X &= A^T\overleftarrow{\xi}_Y \\ \overleftarrow{W}_X &= A^T\overleftarrow{W}_Y A\end{aligned}$$

where $\overleftarrow{\xi}_X \triangleq \overleftarrow{W}_X\overleftarrow{m}_X$ and $\overleftarrow{W}_X \triangleq \overleftarrow{V}_X^{-1}$ (and similarly for Y).

CODE EXAMPLE

Let's calculate the Gaussian forward and backward messages for the addition node in ForneyLab.

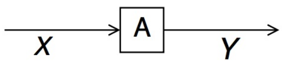



```
# Forward message towards Z
fg = FactorGraph()
@RV x ~ GaussianMeanVariance(constant(1.0), constant(1.0), id=:x)
@RV y ~ GaussianMeanVariance(constant(2.0), constant(1.0), id=:y)
@RV z = x + y; z.id = :z

eval(parse(sumProductAlgorithm(z, name="_z_fwd")))
msg_forward_Z = step_z_fwd!(Dict())[:z]
print("Forward message on Z: $(msg_forward_Z)")

# Backward message towards X
fg = FactorGraph()
@RV x = Variable(id=:x)
@RV y ~ GaussianMeanVariance(constant(2.0), constant(1.0), id=:y)
@RV z = x + y
GaussianMeanVariance(z, constant(3.0), constant(1.0), id=:z)

eval(parse(sumProductAlgorithm(x, name="_x_bwd")))
msg_backward_X = step_x_bwd!(Dict())[:x]
print("Backward message on X: $(msg_backward_X)")
```

```
Forward message on Z: (m=3.00, v=2.00)
Backward message on X: (m=1.00, v=2.00)
```

CODE EXAMPLE

In the same way we can also investigate the forward and backward messages for the gain node




```
# Forward message towards Y
fg = FactorGraph()
@RV x ~ GaussianMeanVariance(constant(1.0), constant(1.0), id=:x)
@RV y = constant(4.0) * x; y.id = :y

eval(parse(sumProductAlgorithm(y, name="_y_fwd")))
msg_forward_Y = step_y_fwd!(Dict())[:y]
print("Forward message on Y: $(msg_forward_Y)")
```

```
Forward message on Y: (m=4.00, v=16.00)
```

```
# Backward message towards X
fg = FactorGraph()
x = Variable(id=:x)
@RV y = constant(4.0) * x
GaussianMeanVariance(y, constant(2.0), constant(1.0))

eval(parse(sumProductAlgorithm(x, name="_x_fwd2")))
msg_backward_X = step_x_fwd2!(Dict())[:x]
print("Backward message on X: $(msg_backward_X)")
```

```
Backward message on X: (m=0.50, v=0.06)
```

Example: Bayesian Linear Regression

- Recall: the goal of regression is to estimate an unknown function from a set of (noisy) function values.

- Assume we want to estimate some function $f : \mathbb{R}^D \to \mathbb{R}$ from data set $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, where $y_i = f(x_i) + \epsilon_i$.

- We will assume a linear model with white Gaussian noise adn a Gaussian prior on the coefficients w:

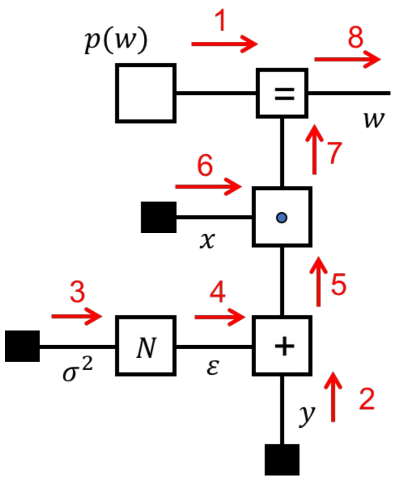
$$
\begin{aligned}
y_i &= w^T x_i + \epsilon_i \\
\epsilon_i &\sim \mathcal{N}(0, \sigma^2) \\
w &\sim \mathcal{N}(0, \Sigma)
\end{aligned}
$$

  or equivalently

$$
\begin{aligned}
p(D, w) &= \overbrace{p(w)}^{\text{weight prior}} \prod_{i=1}^{N} \overbrace{p(y_i \mid x_i, w, \epsilon_i)}^{\text{regression model}} \overbrace{p(\epsilon_i)}^{\text{noise model}} \\
&= \mathcal{N}(w \mid 0, \Sigma) \prod_{i=1}^{N} \delta(y_i - w^T x_i - \epsilon_i) \mathcal{N}(\epsilon_i \mid 0, \sigma^2)
\end{aligned}
$$

- We are interested in inferring the posterior $p(w|D)$

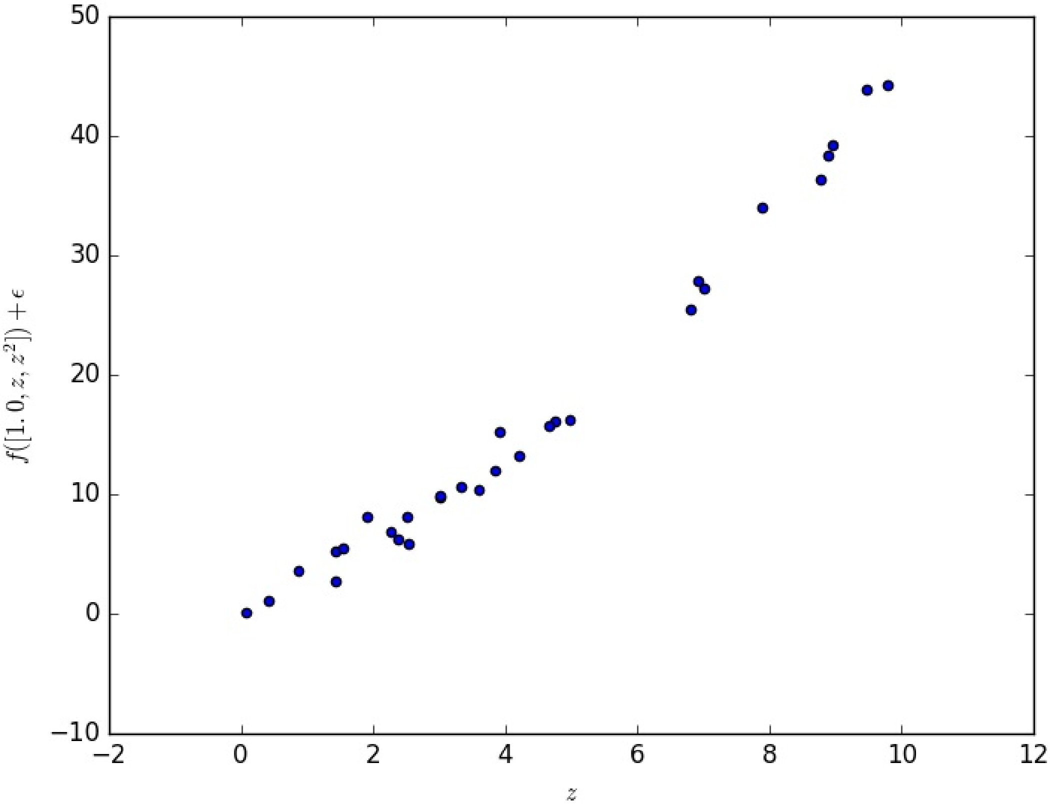
- Here's the factor graph for this model



CODE EXAMPLE

Let's build the factor graph in Julia (with the FFG toolbox ForneyLab)

```julia
using PyPlot
include("scripts/innerproduct_node.jl");

# Parameters
Σ = 1e5 * eye(3) # Covariance matrix of prior on w
σ2 = 2.0         # Noise variance

# Generate data set
w = [1.0; 2.0; 0.25]
N = 30
z = 10.0*rand(N)
x_train = [[1.0; z; z^2] for z in z] # Feature vector x = [1.0; z; z^2]
f(x) = (w'*x)[1]
y_train = map(f, x_train) + sqrt(σ2)*randn(N) # y[i] = w' * x[i] + ϵ
scatter(z, y_train); xlabel(L"z"); ylabel(L"f([1.0, z, z^2]) + \epsilon")
```

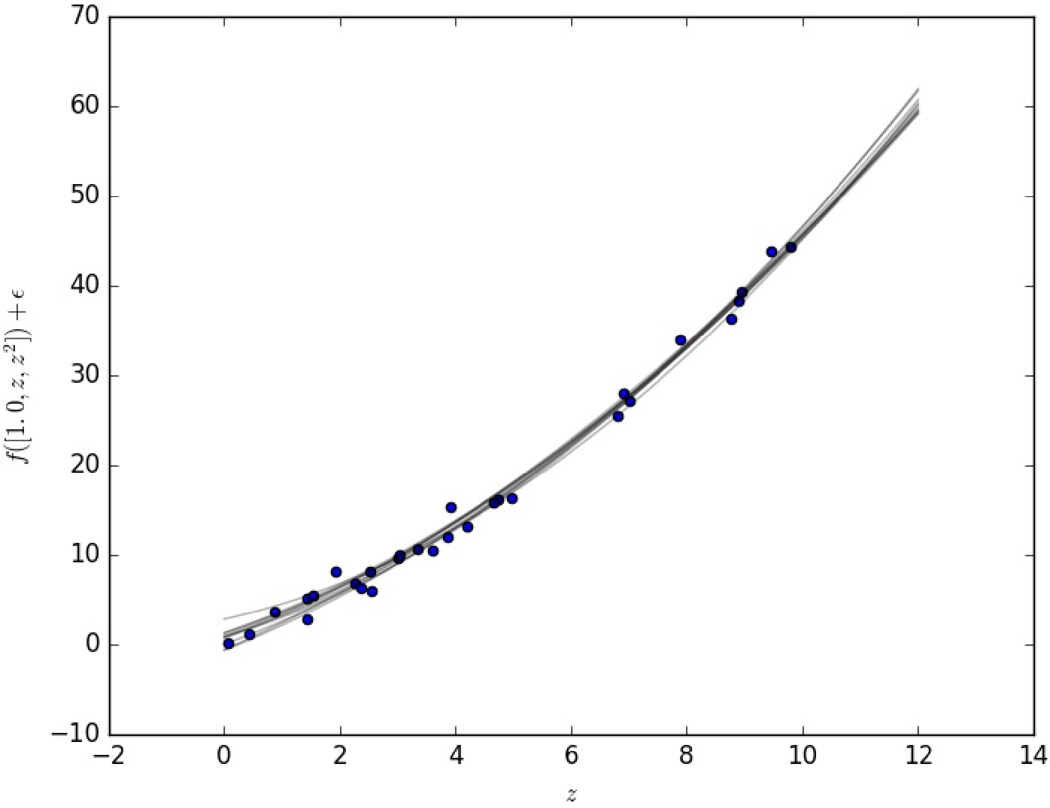



```
PyObject <matplotlib.text.Text object at 0x7f02b1504748>
```

CODE EXAMPLE

Perform sum-product message passing and plot result (mean of posterior)

```
# Build factorgraph
fg = FactorGraph()
@RV w ~ GaussianMeanVariance(constant(zeros(3)), constant(Σ, id=:Σ), id=:w) # p(w)
for t=1:N
    x_t = Variable(id=:x_*t)
    d_t = Variable(id=:d_*t) # d=w'*x
    DotProduct(d_t, x_t, w) # p(f|w,x)
    @RV y_t ~ GaussianMeanVariance(d_t, constant(σ2, id=:σ2_*t), id=:y_*t) # p(y|d)
    placeholder(x_t, :x, index=t, dims=(3,))
    placeholder(y_t, :y, index=t);
end

# Build and run message passing algorithm
eval(parse(sumProductAlgorithm(w)))
data = Dict(:x => x_train, :y => y_train)
w_posterior_dist = step!(data)[:w]

# Plot result
println("Posterior distribution of w: $(w_posterior_dist)")
scatter(z, y_train); xlabel(L"z"); ylabel(L"f([1.0, z, z^2]) + \epsilon");
z_test = collect(0:0.2:12)
x_test = [[1.0; z; z^2] for z in z_test]
for sample=1:10
    w = ForneyLab.sample(w_posterior_dist)
    f_est(x) = (w'*x)[1]
    plot(z_test, map(f_est, x_test), "k-", alpha=0.3);
end
```




```
Posterior distribution of w: (m=[0.74, 2.11, 0.24], v=[[0.59, -0.26, 0.02][-0.26,
0.14, -0.01][0.02, -0.01, 1.27e-03]])
```

Homework Exercises

- (Ex.1) Reflect on the fact that we now have methods for both marginalization and processing observations in FFGs. In principle, we are sufficiently equipped to do inference in probabilistic models through message passing. Draw the graph for
$$p(x_1, x_2, x_3) = f_a(x_1) \cdot f_b(x_1, x_2) \cdot f_c(x_2, x_3)$$
and show which boxes need to be closed for computing $p(x_1|x_2)$.

- (Ex.2) Consider a variable X with measurements $D=\{x_1,x_2\}$. We assume the following model for X:

$$\begin{aligned} p(D,\theta) &= p(\theta)\cdot \prod_{n=1}^{2} p(x_n|\theta) \\ p(\theta) &= \mathcal{N}(\theta \mid 0,1) \\ p(x_n \mid \theta) &= \mathcal{N}(x_n \mid \theta,1) \end{aligned}$$

    - Draw the factor graph and infer θ through the Sum-Product Algorithm.

Inference in Linear Gaussian Models by Sum-Product Message Passing

- The foregoing message update rules can be extended to all scenarios involving additions, fixed-gain multiplications and branching (equality nodes), thus creating a completely **automatable inference framework** for factorized linear Gaussian models.

- The update rules for elementary and important node types can be put in a Table (see **Tables 1 through 6** in Loeliger, 2007).

- If the update rules for all node types in a graph have been tabulated, then inference by message passing comes down to a set of table-lookup operations. This also works for large graphs (where 'manual' inference becomes intractable).

- If the graph contains no cycles, the Sum-Product Algorithm computes **exact** marginals for all hidden variables.

- If the graph contains cycles, we have in principle an infinite tree without terminals. In this case, the SP Algorithm is not guaranteed to find exact marginals. In practice, if we apply the SP algorithm for just a few iterations we often find satisfying approximate marginals.

Dynamic Latent Variable Models

Preliminaries

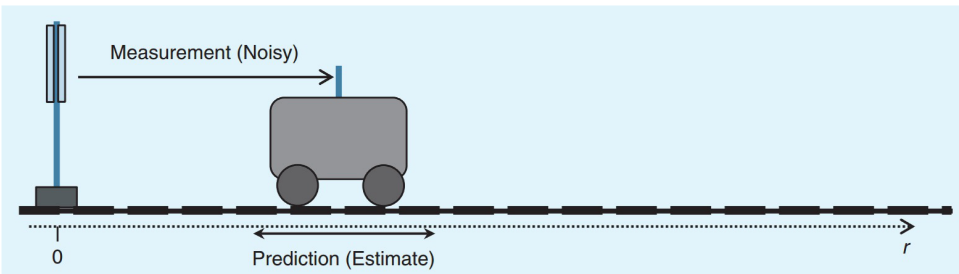
- Goal
  - Introduction to dynamic (=temporal) Latent Variable Models, including the Hidden Markov Model and Kalman filter.
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp.605-615 on Hidden Markov Models
    - Bishop pp.635-641 on Kalman filters
    - Faragher (2012), Understanding the Basis of the Kalman Filter
    - Minka (1999), From Hidden Markov Models to Linear Dynamical Systems

Example Problem

- We consider a one-dimensional cart position tracking problem, see Faragher 2012.
- The hidden states are the position z_t and velocity $\dot{z}_t$. We can apply an external acceleration/breaking force u_t. (Noisy) observations are represented by x_t.
- The equations of motions are given by

$$\begin{aligned} \begin{bmatrix} z_t \\ \dot{z}_t \end{bmatrix} &= \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z_{t-1} \\ \dot{z}_{t-1} \end{bmatrix} + \begin{bmatrix} (\Delta t)^2/2 \\ \Delta t \end{bmatrix} u_t + \mathcal{N}(0, \Sigma_z) \\ x_t &= \begin{bmatrix} z_t \\ \dot{z}_t \end{bmatrix} + \mathcal{N}(0, \Sigma_x) \end{aligned}$$

- Infer the position after 10 time steps.



Dynamical Models

- Consider the *ordered* observation sequence $x^T \triangleq (x_1, x_2, \ldots, x_T)$.

- We wish to develop a generative model

$$p(x^T \mid \theta)$$

  that 'explains' the time series x^T.

- We cannot use the IID assumption $p(x^T|\theta) = \prod_t p(x_t \mid \theta)$. In general, we *can* use the **chain rule** (a.k.a. the general product rule)

$$\begin{aligned} p(x^T) &= p(x_T|x^{T-1})\,p(x^{T-1}) \\ &= p(x_T|x^{T-1})\,p(x_{T-1}|x^{T-2})\cdots p(x_2|x_1)\,p(x_1) \\ &= p(x_1)\prod_{t=2}^{T} p(x_t \mid x^{t-1}) \end{aligned}$$

- Generally, we will want to limit the depth of dependencies on previous observations. For example, the Mth-order linear **Auto-Regressive** (AR) model

$$p(x_t|x^{t-1}) = \mathcal{N}\left(\sum_{m=1}^{M} a_m x_{t-m}\,,\sigma^2\right)$$

limits the dependencies to the past M samples.

State-space Models

- A limitation of AR models is that they need a lot of parameters in order to create a flexible model. E.g., if x_t is an K-dimensional discrete variable, then an Mth-order AR model will have $K^{M-1}(K-1)$ parameters.

- Similar to our work on Gaussian Mixture models and latent Factor models, we can create a flexible dynamic system by introducing *latent* (unobserved) variables $z^T \triangleq (z_1, z_2, \ldots, z_T)$ (one z_t for each observation x_t). In dynamic systems, the latent variables z_t are usually called *state variables*.

- A general **state space model** is defined by

$$\begin{aligned} &p(z_t \mid z^{t-1}) && \text{(state transition model)} \\ &p(x_t \mid z_t) && \text{(observation model)} \\ &p(z_1) && \text{(initial state)} \end{aligned}$$

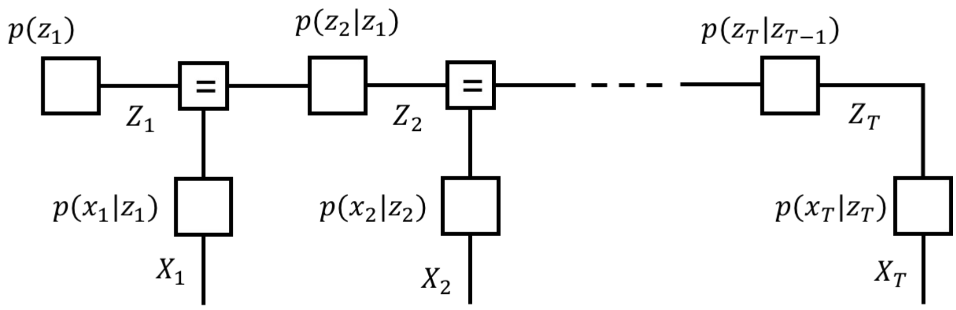
- A very common computational assumption is to let state transitions be ruled by a *first-order Markov chain* as

$$p(z_t \mid z^{t-1}) = p(z_t \mid z_{t-1})$$

- The Markov assumption leads to the following joint probability distribution for the state-space model:

$$p(x^T, z^T) = \underbrace{p(z_1)}_{\text{initial state}} \prod_{t=2}^{T} \underbrace{p(z_t \mid z_{t-1})}_{\text{state transitions}} \prod_{t=1}^{T} \underbrace{p(x_t \mid z_t)}_{\text{observations}}$$

- The Forney-style factor graph for a state-space model:



- Exercise: Show that in a state-space model x_t is not a first-order Markov chain in the observations, i.e., show that

$$p(x_t \mid x_{t-1}, x_{t-2}) \neq p(x_t \mid x_{t-1})\,.$$

Hidden Markov Models and Linear Dynamical Systems

- A **Hidden Markov Model** (HMM) is a state-space model with discrete-valued state variables Z_t.

- E.g., Z_t is a K-dimensional hidden binary 'class indicator' with transition probabilities $A_{jk} \triangleq p(z_{tk} = 1 \mid z_{t-1,j} = 1)$, or equivalently
$$p(z_t|z_{t-1}) = \prod_{k=1}^{K}\prod_{j=1}^{K} A_{jk}^{z_{t-1,j}z_{tk}}$$
which is usually accompanied by an initial state distribution $\pi_k \triangleq p(z_{1k} = 1)$.

- The classical HMM has also discrete-valued observations but in pratice any (probabilistic) observation model $p(x_t|z_t)$ may be coupled to the hidden Markov chain.

- Another well-known state-space model with continuous-valued state variables Z_t is the **(Linear) Gaussian Dynamical System** (LGDS), which is defined as
$$\begin{aligned}
p(z_t \mid z_{t-1}) &= \mathcal{N}\left(A z_{t-1}\,,\,\Sigma_z\right) \\
p(x_t \mid z_t) &= \mathcal{N}\left(C z_t\,,\,\Sigma_x\right) \\
p(z_1) &= \mathcal{N}\left(\mu_1\,,\,\Sigma_1\right)
\end{aligned}$$

- Note that the joint distribution over $\{(x_1,z_1),\ldots,(x_t,z_t)\}$ is a (large-dimensional) Gaussian distribution. This means that, in principle, every inference problem on the LGDS model also leads to a Gaussian distribution.

- HMM's and LGDS's (and variants thereof) are at the basis of a wide range of complex information processing systems, such as speech and language recognition, robotics and automatic car navigation, and even processing of DNA sequences.

Kalman Filtering

- Technically, a **Kalman filter** is the solution to the recursive estimation (inference) of the hidden state z_t based on past observations in an LGDS, i.e., Kalman filtering solves the problem $p(z_t \mid x^t)$ based on the previous estimate $p(z_{t-1} \mid x^{t-1})$ and a new observation x_t (in the context of the given model specification of course).

- Let's infer the Kalman filter for a scalar linear Gaussian dynamical system:
$$\begin{aligned}
p(z_t \mid z_{t-1}) &= \mathcal{N}(z_t \mid a z_{t-1}, \sigma_z^2) \qquad &\text{(state transition)} \\
p(x_t \mid z_t) &= \mathcal{N}(x_t \mid c z_t, \sigma_x^2) \qquad &\text{(observation)}
\end{aligned}$$

- Kalman filtering comprises inferring $p(z_t \mid x^t)$ from a given prior estimate $p(z_{t-1} \mid x^{t-1})$ and a new observation x_t. Let us assume that
$$p(z_{t-1} \mid x^{t-1}) = \mathcal{N}(z_{t-1} \mid \mu_{t-1}, \sigma_{t-1}^2) \qquad \text{(prior)}$$

- Note that everything is Gaussian, so this is *in principle* possible to execute inference problems analytically and the result will be a Gaussian posterior:

$$
\begin{aligned}
p(z_t \mid x^t) &= p(z_t \mid x_t, x^{t-1}) \propto p(x_t, z_t \mid x^{t-1}) \\
&\propto p(x_t \mid z_t)\, p(z_t \mid x^{t-1}) \\
&= p(x_t \mid z_t) \sum_{z_{t-1}} p(z_t, z_{t-1} \mid x^{t-1}) \\
&= \underbrace{p(x_t \mid z_t)}_{\text{observation}} \sum_{z_{t-1}} \underbrace{p(z_t \mid z_{t-1})}_{\text{state transition}} \underbrace{p(z_{t-1} \mid x^{t-1})}_{\text{prior}} \\
&= \mathcal{N}(x_t \mid c z_t, \sigma_x^2) \sum_{z_{t-1}} \mathcal{N}(z_t \mid a z_{t-1}, \sigma_z^2)\, \mathcal{N}(z_{t-1} \mid \mu_{t-1}, \sigma_{t-1}^2) \\
&\propto \mathcal{N}\left(z_t \,\middle|\, \frac{x_t}{c}, \left(\frac{\sigma_x}{c}\right)^2\right) \times \mathcal{N}\left(z_t \,\middle|\, a\mu_{t-1}, \sigma_z^2 + (a\sigma_{t-1})^2\right) \\
&= \mathcal{N}\left(z_t \mid \mu_t, \sigma_t^2\right)
\end{aligned}
$$

with

$$
\begin{aligned}
\rho_t^2 &= \sigma_z^2 + a^2 \sigma_{t-1}^2 && \text{(auxiliary variable)} \\
K_t &= \frac{c\rho_t^2}{c^2 \rho_t^2 + \sigma_x^2} && \text{('Kalman gain')} \\
\mu_t &= a\mu_{t-1} + K_t \cdot (x_t - c a \mu_{t-1}) && \text{(posterior mean)} \\
\sigma_t^2 &= (1 - K_t)\, \rho_t^2 && \text{(posterior variance)}
\end{aligned}
$$

- Kalman filtering consists of computing these four equations for each new observation (x_t).

- In short, it is possible to analytically derive the Kalman filter for a linear dynamical system with Gaussian state and observation noise (eventhough it is not a fun exercise).

- If anything changes in the model, e.g., the state noise is not Gaussian, then you have to re-derive the inference equations again from scratch and it may not lead to an analytically pleasing answer.

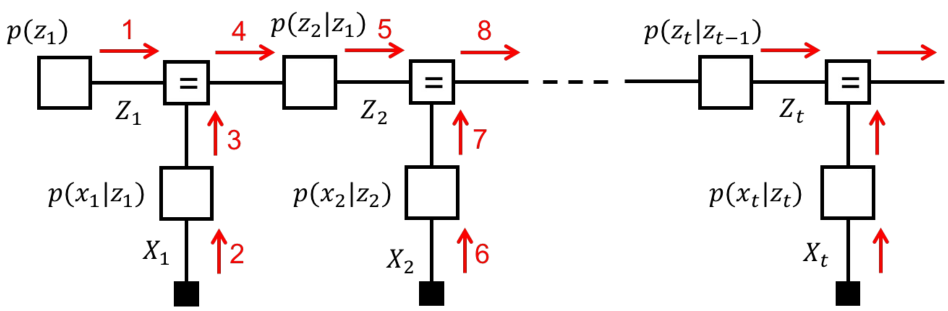
- $\Rightarrow$ Generally, we will want to automate the inference process.

Message Passing in State-space Models

- Once the (state-space) models have been specified, we can define state and parameter estimation problems as inference tasks on the generative model.

- In principle, for linear Gaussian models these inference tasks can be analytically solved, see e.g. Faragher, 2012
  - These derivations quickly become quite laborious

- Alternatively, we could specify the generative model in a (Forney-style) factor graph and use automated message passing to infer the posterior over the hidden variables. E.g., the message passing schedule for Kalman filtering looks like this:



Example Problem Revisited

We can solve the cart tracking problem by sum-product message passing in a factor graph like the one depicted above. All we have to do is create factor nodes for the state-transition model $p(z_t|z_{t-1})$ and the observation model $p(x_t|z_t)$. Then we just build the factor graph and let ForneyLab (factor graph toolbox) perform message passing.

We'll implement the following model:

$$\begin{aligned}
\begin{bmatrix} z_t \\ \dot{z}_t \end{bmatrix} &= \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z_{t-1} \\ \dot{z}_{t-1} \end{bmatrix} + \begin{bmatrix} (\Delta t)^2/2 \\ \Delta t \end{bmatrix} u_t + \mathcal{N}(0, \Sigma_z) \\
\mathbf{x}_t &= \begin{bmatrix} z_t \\ \dot{z}_t \end{bmatrix} + \mathcal{N}(0, \Sigma_x)
\end{aligned}$$

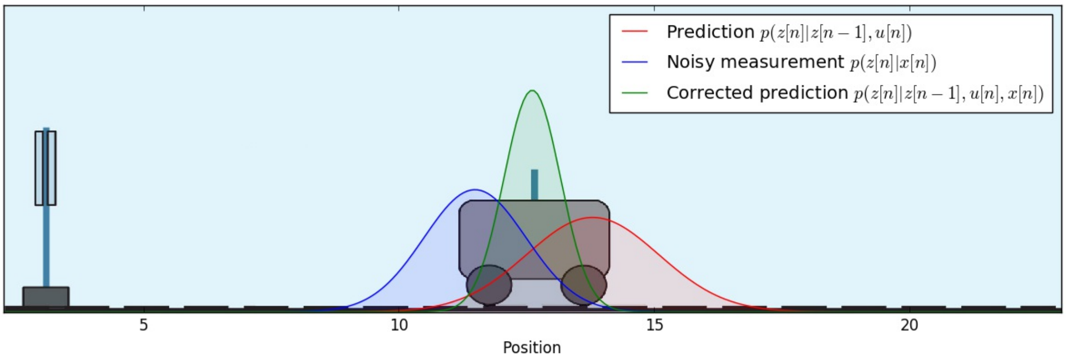
```julia
using ForneyLab
include("scripts/cart_tracking_helpers.jl") # implements required factor nodes + helper functions

# Specify the model parameters
Δt = 1.0                     # assume the time steps to be equal in size
A = [1.0 Δt;
     0.0 1.0]
b = [0.5*Δt^2; Δt]
Σz = diagm([0.2*Δt; 0.1*Δt]) # process noise covariance
Σx = diagm([1.0; 2.0])      # observation noise covariance;

# Generate noisy observations
n = 10                # perform 10 timesteps
z_start = [10.0; 2.0] # initial state
u = 0.2 * ones(n)     # constant input u
noisy_x = generateNoisyMeasurements(z_start, u, A, b, Σz, Σx);
```

Since the factor graph is just a concatination of n identical "sections", we only have to specify a single section. When running the message passing algorithm we will explictly use the posterior of the previous timestep as prior in the next one. Let's build a section of the factor graph:

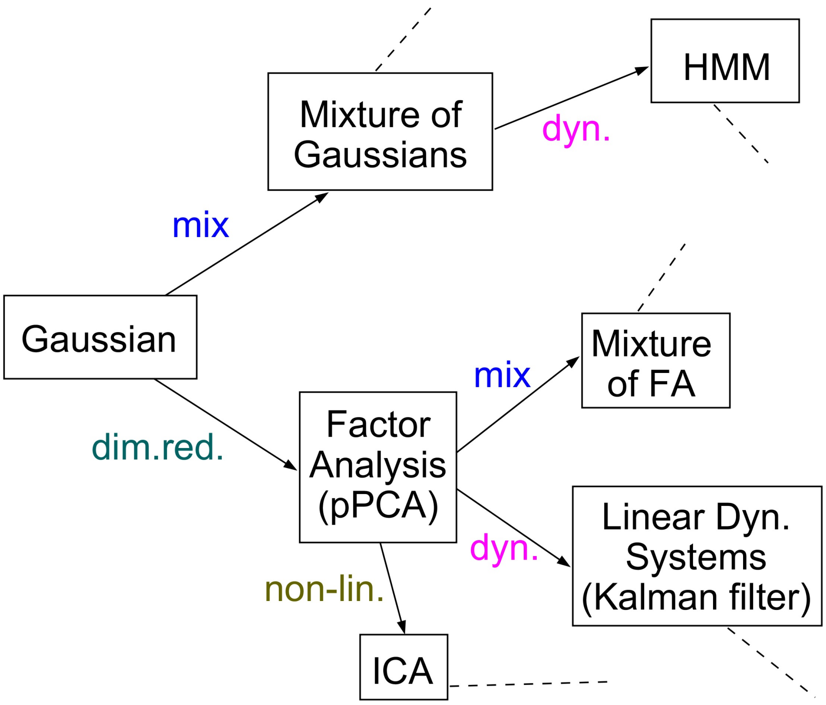
```julia
fg = FactorGraph()
z_prev_m = Variable(id=:z_prev_m); placeholder(z_prev_m, :z_prev_m, dims=(2,))
z_prev_v = Variable(id=:z_prev_v); placeholder(z_prev_v, :z_prev_v, dims=(2,2))
bu = Variable(id=:bu); placeholder(bu, :bu, dims=(2,))

@RV z_prev ~ GaussianMeanVariance(z_prev_m, z_prev_v, id=:z_prev) # p(z_prev)
@RV noise_z ~ GaussianMeanVariance(constant(zeros(2), id=:noise_z_m), constant(Σz, id=:noise_z_v))
 # process noise
@RV z = constant(A, id=:A) * z_prev + bu + noise_z; z.id = :z # p(z|z_prev) (state transition mode
l)
@RV x ~ GaussianMeanVariance(z, constant(Σx, id=:Σx)) # p(x|z) (observation model)
placeholder(x, :x, dims=(2,));
# ForneyLab.draw(fg)
```

Now that we've built the factor graph, we can perform Kalman filtering by inserting measurement data into the factor graph and performing message passing.

```
eval(parse(sumProductAlgorithm(z))) # build message passing algorithm
marginals = Dict()
messages = Array{Message}(6)
data = Dict(:x => noisy_x[1], :bu => b*u[1], :z_prev_m => zeros(2), :z_prev_v => 1e8*eye(2))
for t=1:n
    step!(data, marginals, messages) # perform msg passing (single timestep)
    # Posterior of z becomes prior of z in the next timestep:
    ForneyLab.ensureParameters!(marginals[:z], (:m, :v))
    data[:z_prev_m] = marginals[:z].params[:m]
    data[:z_prev_v] = marginals[:z].params[:v]
end

# Collect prediction p(z[n]|z[n-1]), measurement p(z[n]|x[n]), corrected prediction p(z[n]|z[n-1], x[n])
prediction     = messages[5].dist # the message index is found by manual inspection of the schedule
measurement    = messages[6].dist
corr_prediction = marginals[:z]

# Make a fancy plot of the prediction, noisy measurement, and corrected prediction after n timesteps
plotCartPrediction(prediction, measurement, corr_prediction);
```



Extensions



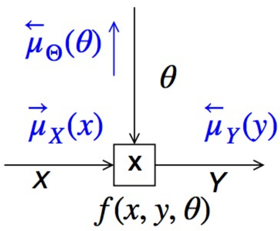
EM as a Message Passing Algorithm

Preliminaries

- Goals
  - Describe Expectation-Maximization (EM) as a message passing algorithm on a Forney-style factor graph
- Materials
  - Madatory
    - These lecture notes
  - Optional
    - Dauwels et al., 2009, pp. 1-5 (sections I,II and III)

A Problem for the Multiplier Node

- Consider the multiplier factor $f(x,y,\theta) = \delta(y-\theta x)$ with incoming Gaussian messages $\overrightarrow{\mu}_X(x) = \mathcal{N}(x|m_x,v_x)$ and $\overleftarrow{\mu}_Y(y) = \mathcal{N}(y|m_y,v_y)$. For simplicity's sake, we assume all variables are scalar.



- In a system identification setting, we are interested in computing the outgoing message $\overleftarrow{\mu}_\Theta(\theta)$.
- Let's compute the sum-product message:

$$
\begin{aligned}
\overleftarrow{\mu}_\Theta(\theta) &= \int \overrightarrow{\mu}_X(x)\, \overleftarrow{\mu}_Y(y)\, f(x,y,\theta)\, \mathrm{d}x\mathrm{d}y \\
&= \int \mathcal{N}(x\,|\,m_x,v_x)\, \mathcal{N}(y\,|\,m_y,v_y)\, \delta(y-\theta x)\; \mathrm{d}x\mathrm{d}y \\
&= \int \mathcal{N}(x\,|\,m_x,v_x)\, \mathcal{N}(\theta x\,|\,m_y,v_y)\, \mathrm{d}x \\
&= \int \mathcal{N}(x\,|\,m_x,v_x)\, \mathcal{N}\left(x \,\middle|\, \frac{m_y}{\theta}, \frac{v_y}{\theta^2}\right)\, \mathrm{d}x \\
&= \mathcal{N}\left(\frac{m_y}{\theta} \,\middle|\, m_x, v_x + \frac{v_y}{\theta^2}\right) \cdot \int \mathcal{N}(x\,|\,m_*,v_*)\, \mathrm{d}x \qquad \text{(SRG-6)} \\
&= \mathcal{N}\left(\frac{m_y}{\theta} \,\middle|\, m_x, v_x + \frac{v_y}{\theta^2}\right)
\end{aligned}
$$

- This is **not** a Gaussian message for Θ! Passing this message into the graph leads to very serious problems when trying to compute sum-product messages for other factors in the graph.
  - (We have seen before in the lesson on Working with Gaussians that multiplication of two Gaussian-distributed variables does *not* produce a Gaussian distributed variable.)
- The same problem occurs in a forward message passing schedule when we try to compute a message for Y from incoming Gaussian messages for both X and Θ.

Limitations of Sum-Product Messages

- The foregoing example shows that the sum-product (SP) message update rule will sometimes not do the job. For example:
  - On large-dimensional **discrete** domains, the SP update rule maybe computationally intractable.
  - On **continuous** domains, the SP update rule may not have a closed-form solution or the rule may lead to a function that is incompatible with Gaussian message passing.
- There are various ways to cope with 'intractable' SP update rules. In this lesson, we discuss how the EM-algorithm can be written as a message passing algorithm on factor graphs. Then, we will solve the 'multiplier node problem' with EM messages (rather than with SP messages).

EM as Message Passing

- Consider first a general setting with likelihood function $f(x,\theta)$, hidden variables x and tuning parameters θ. Assume that we are interested in the maximum likelihood estimate

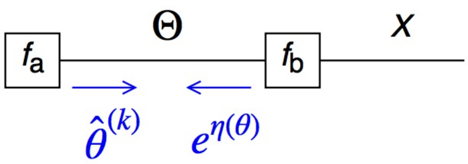
$$\hat{\theta} = \arg\max_\theta \int f(x,\theta)\mathrm{d}x \,.$$

- If $\int f(x,\theta)\mathrm{d}x$ is intractible, we can try to apply the EM-algorithm to estimate $\hat{\theta}$, which leads to the following iterations (*cf.* lesson on the EM algorithm):

$$\hat{\theta}^{(k+1)} = \underbrace{\arg\max_\theta}_{\text{M-step}} \left(\underbrace{\int_x f(x,\hat{\theta}^{(k)})\,\log f(x,\theta)\,\mathrm{d}x}_{\text{E-step}} \right)$$

- It turns out that *for factorized functions* $f(x,\theta)$, the EM-algorihm can be executed as a message passing algorithm on the factor graph.
- As an simple example, we consider the factorization

$$f(x,\theta) = f_a(\theta)\,f_b(x,\theta)$$



- Applying the EM-algorithm to this graph leads to the following forward and backward messages over the θ edge

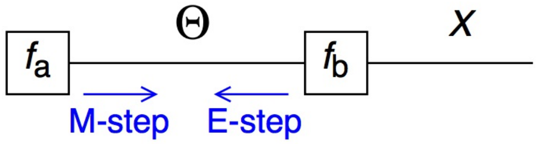
$$\begin{aligned}
\textbf{E-step}: &\quad \eta(\theta) = \int p_b(x|\hat{\theta}^{(k)}) \log f_b(x,\theta)\,\mathrm{d}x \\
\textbf{M-step}: &\quad \hat{\theta}^{(k+1)} = \arg\max_\theta \left(f_a(\theta)\, e^{\eta(\theta)} \right)
\end{aligned}$$

where $p_b(x|\hat{\theta}^{(k)}) \triangleq \frac{f_b(x,\hat{\theta}^{(k)})}{\int f_b(x',\hat{\theta}^{(k)})\,\mathrm{d}x'}$.

Proof:

$$
\begin{aligned}
\hat{\theta}^{(k+1)} &= \arg\max_{\theta} \int_x f(x, \ddot{\theta}^{(k)}) \, \log f(x,\theta) \, \mathrm{d}x \\
&= \arg\max_{\theta} \int_x f_a(\theta) f_b(x, \hat{\theta}^{(k)}) \, \log(f_a(\theta) f_b(x,\theta)) \, \mathrm{d}x \\
&= \arg\max_{\theta} \int_x f_b(x, \hat{\theta}^{(k)}) \cdot (\log f_a(\theta) + \log f_b(x,\theta)) \, \mathrm{d}x \\
&= \arg\max_{\theta} \left(\log f_a(\theta) + \frac{\int f_b(x, \hat{\theta}^{(k)}) \log f_b(x,\theta) \, \mathrm{d}x}{\int f_b(x', \hat{\theta}^{(k)}) \, \mathrm{d}x'} \right) \\
&= \arg\max_{\theta} \left(\log f_a(\theta) + \underbrace{\int p_b(x|\ddot{\theta}^{(k)}) \log f_b(x,\theta) \, \mathrm{d}x}_{\eta(\theta)} \right) \\
&= \underbrace{\arg\max_{\theta}}_{\text{M-step}} \left(f_a(\theta) \underbrace{e^{\eta(\theta)}}_{\text{E-step}} \right)
\end{aligned}
$$

- The messages represent the 'E' and 'M' steps, respectively:



- The quantity $\eta(\theta)$ (a.k.a. the **E-log message**) may be interpreted as a log-domain summary of f_b. The message $e^{\eta(\theta)}$ is the corresponding 'probability domain' message that is consistent with the semantics of messages as summaries of factors. In a software implementation, you can use either domain, as long as a consistent method is chosen.

- Note that the denominator $\int f_b(x', \ddot{\theta}^{(k)}) \, \mathrm{d}x'$ in p_b is just a scaling factor that can usually be ignored, leading to a simpler E-log message

$$
\eta(\theta) = \int f_b(x, \hat{\theta}^{(k)}) \log f_b(x,\theta) \, \mathrm{d}x \, .
$$

EM vs SP and MP Message Passing

- Consifer again the likelihood model $f(x,\theta)$ with x a set of hidden variables. We are interested in the ML estimate

$$\hat{\theta} = \arg\max_\theta \int f(x,\theta)\mathrm{d}x\,.$$

- Recall that in a 'regular' (*not* message passing) setting, the EM-algorithm is particularly useful when the *expectation* (E-step)

$$\eta(\theta) = \int_x f(x,\hat{\theta}^{(k)})\, \log f(x,\theta)\, \mathrm{d}x$$

  leads to easier expressions than the *marginalization* (which is what we really want)

$$\bar{f}(\theta) = \int f(x,\theta)\mathrm{d}x.$$

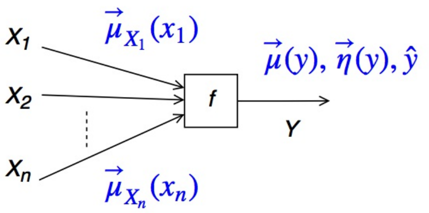
- Similarly, in a message passing framework with connected nodes f_a and f_b, EM messages are particularly useful when the *expectation* (represented by the **E-log message**)

$$\eta(\theta) = \int f_b(x|\hat{\theta}^{(k)}) \log f_b(x,\theta)\, \mathrm{d}x$$

  leads to easier expressions than the *marginalization* (represented by the **sum-product message**, which is also what we really want)

$$\mu(\theta) = \int f_b(x,\theta)\mathrm{d}x.$$

- Just as for the sum-product (SP) and max-product (MP) messages, we can work out the outgoing E-log message on the Y edge for a *general* node $f(x_1,\ldots,x_M,y)$ with given message inputs $\overrightarrow{\mu}_{X_m}(x_m)$ (see also Dauwels et al. (2009), Table-1, pg.4):



$$\textbf{SP}: \quad \overrightarrow{\mu}(y) = \int \overrightarrow{\mu}_{X_1}(x_1)\cdots \overrightarrow{\mu}_{X_M}(x_M)\, f(x_1,\ldots,x_M,y)\, \mathrm{d}x_1\ldots\mathrm{d}x_M$$

$$\textbf{MP}: \quad \hat{y} = \arg\max_{x_1,\ldots,x_M} \overrightarrow{\mu}_{X_1}(x_1)\cdots \overrightarrow{\mu}_{X_M}(x_M)\, f(x_1,\ldots,x_M,y)$$

$$\textbf{E-log}: \quad \overrightarrow{\eta}(y) = \int p(x_1,\ldots,x_M|y^{(k)})\, \log f(x_1,\ldots,x_M,y)\, \mathrm{d}x_1\ldots\mathrm{d}x_M$$

where $p(x_1,\ldots,x_M|y^{(k)}) \triangleq \frac{\overrightarrow{\mu}_{X_1}(x_1)\cdots\overrightarrow{\mu}_{X_M}(x_M)\, f(x_1,\ldots,x_M,\hat{y}^{(k)})}{\int \overrightarrow{\mu}_{X_1}(x_1)\cdots\overrightarrow{\mu}_{X_M}(x_M)\, f(x_1,\ldots,x_M,\hat{y}^{(k)})\, \mathrm{d}x_1\ldots\mathrm{d}x_M}.$

- **Exercise**: proof the generic E-log message update rule.

A Snag for EM Message Passing on Deterministic Nodes

- The factors for deterministic nodes are (Dirac) delta functions, e.g., $\delta(y-\theta x)$ for the multiplier.

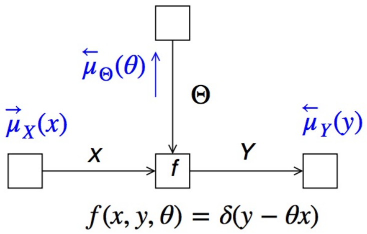
- Note that the outgoing E-log message for a deterministic node will also be a delta function, since the expectation of $\log\delta(\cdot)$ is again a delta function. For details, consult Dauwels et al. (2009) pg.5, section F.

- This would stall the iterative estimation process at the current estimate since the outgoing E-log message would express complete certainty about the estimate.

- This issue can be resolved by closing a box around a subgraph that includes (the deterministic node) f and *at least one non-deterministic factor*. EM message passing can now proceed with the newly created node.

A Solution for the Multiplier Node with Unknown Coefficient

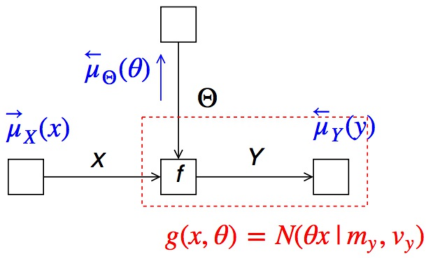
- We get back no to the original problem in this lesson. Consider again the (scalar) multiplier with unknown coefficient $f(x,y,\theta) = \delta(y-\theta x)$ and incoming messages $\overrightarrow{\mu_X}(x) = \mathcal{N}(x|m_x, v_x)$ and $\overleftarrow{\mu_Y}(y) = \mathcal{N}(y|m_y, v_y)$.



We will now compute the outgoing E-log message for Θ.

- Since $f(x,y,\theta)$ is deterministic, we will first group f with the (non-deterministic) node $\overleftarrow{\mu_Y}(y) = \mathcal{N}(y|m_y, v_y)$, leading (through sum-product rule) to

$$
\begin{aligned}
g(x,\theta) &\triangleq \int \overleftarrow{\mu_Y}(y)\, f(x,y,\theta)\,\mathrm{d}y \\
&= \int \mathcal{N}(y|m_y,v_y)\,\delta(y-\theta x)\,\mathrm{d}y \\
&= \mathcal{N}(\theta x \,|\, m_y, v_y)\,.
\end{aligned}
$$



- The problem now is to pass an E-log message out of $g(x,\theta)$. Assume that g has received an estimate $\hat{\theta}$ from the incoming message over the Θ edge. The E-log update rule then prescribes

$$
\begin{aligned}
\eta(\theta) &= \mathbb{E}\left[\log g(x,\theta)\right] \\
&= \mathbb{E}\left[\mathcal{N}(\theta x|m_y,v_y)\right] \\
&= \text{const.} - \frac{1}{2v_y}\left(\mathbb{E}[X^2]\theta^2 - 2m_y\mathbb{E}[X]\theta + m_y^2\right) \\
&\propto \mathcal{N}_\xi\left(\theta \,\middle|\, \frac{m_y\mathbb{E}[X]}{v_y}, \frac{\mathbb{E}\left[X^2\right]}{v_y}\right)
\end{aligned}
$$

where we used the 'canonical' parametrization of the Gaussian $\mathcal{N}_\xi(\theta \mid \xi, w) \propto \exp\left(\xi\theta - \frac{1}{2}w\theta^2\right)$.

- In the E-log message update rule, the expections $\mathbb{E}[X]$ and $\mathbb{E}\left[X^2\right]$ have to be taken w.r.t. $p(x|\hat{\theta}) = \overrightarrow{\mu_X}(x)\, g(x,\hat{\theta})$ (consult the generic E-log update rule). A straightforward (but rather painful) derivation leads to

$$
\begin{aligned}
p(x\,|\,\hat{\theta}) &= \overrightarrow{\mu_X}(x)\, g(x,\hat{\theta}) \\
&= \mathcal{N}(x \mid m_x, v_x) \cdot \mathcal{N}(\hat{\theta}x \mid m_y, v_y) \\
&= \mathcal{N}(x \mid m_x, v_x) \cdot \mathcal{N}\left(x \,\middle|\, \frac{m_y}{\hat{\theta}}, \frac{v_y}{\hat{\theta}^2}\right) \\
&\propto \mathcal{N}_\xi(x \mid \xi_g, w_g)
\end{aligned}
$$

where $w_g = \frac{1}{v_x} + \frac{\hat{\theta}^2}{v_y}$ and $\xi_g \triangleq w_g m_g = \frac{m_x}{v_x} + \frac{\hat{\theta}\, m_y}{v_y}$. It follows that

$$
\begin{aligned}
\mathbb{E}[X] &= m_g \\
\mathbb{E}\left[X^2\right] &= m_g^2 + w_g^{-1}
\end{aligned}
$$

- $\Rightarrow$ The E-log update formula may not be fun to derive, but the result is very pleasing: the **E-log message for the multiplier with unknown coefficient is a Gaussian message** with closed-form expressions for its parameters! See also Dauwels et al. (2009) Table-2, pg.6.

Automating Inference

- It follows that, for a dynamical system with unknown coefficients, both state estimation and parameter learning can be achieved through Gaussian message passing based on SP and EM message update rules.
- These (SP and EM) message update rules can be tabularized and implemented in software for a large set of factors that are common in probabilistic models. (See the tables in Loeliger et al. (2007) and Dauwels et al. (2009)).
- Tabulated SP and EM messages for frequently occuring factors facilitate the automated derivation of nontrivial inference algorithms.
- This makes it possible to automate inference for state and parameter estimation in very complex probabilistic model. Here (in the SPS group at TU/e), we are developing such a factor graph toolbox in Julia.
- There is lots more to say about factor graphs. This is a very exciting area of research that promises both
    1. to consolidate a wide range of signal processing and machine learning algorithms in one elegant framework
    2. to automate inference and learning in new models that have previously been untractable for existing machine learning methods.

Example: Linear Dynamical Systems

As before let us consider the linear dynamical system (LDS)

$$
\begin{aligned}
z_n &= A z_{n-1} + w_n \\
x_n &= C z_n + v_n \\
w_n &\sim \mathcal{N}(0,\Sigma_w) \\
v_n &\sim \mathcal{N}(0,\Sigma_v)
\end{aligned}
$$

Again, we will consider the case where x_n is observed and z_n is a hidden state. C , Σ_w and Σ_v are given parameters but in contrast to the previous section, we will assume that the value of parameter A is unknown.